EOARD

SPC - 98 - 4061

Lattice Boltzmann Models for Multicomponent Fluids

Contract No. F61775 - 98 - WE101

item #003 - Final Report

Delivery date: July 20, 1999

Principal investigator

Dr. Victor SOFONEA

Polytechnical University of Timişoara

Romania

DTIC QUALITY INSPECTED 4

Approved for Public Release
Distribution Unlimited

EOARD

SPC - 98 - 4061

Lattice Boltzmann Models for Multicomponent Fluids

Contract No. F61775 - 98 - WE101

item #003 - Final Report

Delivery date: July 20, 1999

Principal investigator
Dr. Victor SOFONEA
Polytechnical University of Timişoara
Romania

19990903 074

AQF99-11-2178

gathering and maintaining the data needed, and collection of information, including suggestions Davis Highway, Suite 1204, Arlington, VA 22202	d completing and reviewing the collection of for reducing this burden to Washington Hea	information. Send comments regard adquarters Services. Directorate for	rding this burden estimate or any other aspect of this Information Operations and Reports, 1215 Jefferson
AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND	DATES COVERED
	20 Jul 1999		Final Report
4. TITLE AND SUBTITLE			5. FUNDING NUMBERS
Lattice Boltzmann Models for N	Multicomponent Fluids		F61775-98-WE101
6. AUTHOR(S)			
Dr. Victor Sofonea			
7. PERFORMING ORGANIZATION NAM	ME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER
Politehnica University of Timiso Bd. Mihai Viteazul 1 Timisoara 1900 Romania	N/A		
		**	
9. SPONSORING/MONITORING AGENC	CY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
EOARD PSC 802 BOX 14 FPO 09499-0200		Ģ.	SPC 98-4061
11. SUPPLEMENTARY NOTES		· · · · · · · · · · · · · · · · · · ·	
12a. DISTRIBUTION/AVAILABILITY STA	TEMENT		12b. DISTRIBUTION CODE
Approved for public release; di	A		
Boltzmann methods in the mo geometry of which is not rela benchmarking against existin component fluid flow in two interactions will be added to a	deling of complex (multicomponent) fluted to the thermal velocities of fluid pg Lattice Boltzmann models will be spatial dimensions, followed by study	uids. A two-dimensional Lattice particles. Comprehensive testin accomplished. Initial testing by of a two-component flow i en multiphase fluids. Two spe	ntractor will investigate the utility of Lattice e Boltzmann model will be developed, the ng and analysis of the model, as well as will be accomplished by modeling one-in two dimensions. Finally, interparticle ecific physical problems investigated: the rface tension on contact angle.
·			
14. SUBJECT TERMS			15. NUMBER OF PAGES
			226
EOARD, Fluid Dynamics, Latti	ce Boltzmann Methods, Computationa	I Mathematics	16. PRICE CODE N/A
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19, SECURITY CLASSIFICAT OF ABSTRACT	TION 20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL
NSN 7540-01-280-5500			Standard Form 298 (Rev. 2-89)

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Contents

1	Fini	te Difference Schemes for the Boltzmann Equation
	1.1	Introduction
	1.2	Spatial derivatives on a square lattice
	1.3	Characteristics
	1.4	Interpolation procedures on the characteristics line 8
	1.5	Upwind scheme
	1.6	Lax - Friedrichs scheme
	1.7	Space centered scheme
	1.8	Lax - Wendroff scheme
	1.9	Interpolation Supplemented Schemes
		1.9.1 General Description
		1.9.2 Interpolation procedures on the $2D$ lattice 17
	1.10	Numerical simulations
		1.10.1 Computer code
		1.10.2 Single Component Fluid
		1.10.3 Homogeneous Two Component Fluid
2	Diff	usion Couple 45
	2.1	Description of the model
	2.2	Chapman - Enskog expansion
	2.3	Conservation equations
	2.4	Mass equations
	2.5	Equilibrium velocity \mathbf{u}'
	2.6	Momentum equation
	2.7	Pure diffusion
	2.8	Comparison with the theory of Shan and Doolen 57
	2.9	Simulation results

3	Sess	ile drop																			74
	3.1	Drop shape																			74
	3.2	Contact ang	le		•		•	•	•	•	•	•	•	•	•		•		•	•	7 5
Re	efere	nces																			90
Aj	ppen	dices																			93
A	wet	9 code																			93
В	dif9	code																			176
\mathbf{C}	sw e	code																			209

Chapter 1

Finite Difference Schemes for the Boltzmann Equation

1.1 Introduction

In the intermediate report [1], Centered Space (CS) finite difference schemes were developed to solve the Lattice Boltzmann (LB) evolution equations for the distribution functions $f_i(\mathbf{r},t)$ of an one component fluid system, when the number of non-vanishing discrete velocities was N=6 (the so-called seven bit model) or N=8 (the so-called nine bit model):

$$f_{i}(\mathbf{r}, t + \delta t) \simeq f_{i}(\mathbf{r}, t) - \frac{\delta t}{\tau_{p}} \left[f_{i}(\mathbf{r}, t) - f_{i}^{eq}(\mathbf{r}, t) \right] - \delta t \, \mathbf{e}_{i} \cdot \nabla_{\mathbf{r}} f_{i}(\mathbf{r}, t) + \frac{\delta t}{k_{B}T} \, \mathbf{F}(\mathbf{r}, t) \cdot \left[\mathbf{e}_{i} - \mathbf{u}(\mathbf{r}, t) \right] f_{i}^{eq}(\mathbf{r}, t)$$

$$i = 0, 1, \dots N$$

$$(1.1)$$

In the above equation, as well as in the whole report to follow, there is no implicit summation over the index i. We will use further the implicit summation rule (i.e., summation over repeated indices) only for cartesian components denoted by Greek letters α , β , γ ..., while summation over other indices (including i) will be always explicited, when such operations are present.

The validity of the CS finite difference schemes was successfully tested for two kinds of 2D flows between parallel plates: Poisseuille flow and Couette flow. The correct velocity profiles in the stationary regime (the parabolic

one, for Poisseuille flow, and the linear one, for Couette flow), as well as the right viscosity value were recovered for N=6, as well as for N=8 [1].

Our early attempts to use the CS scheme to simulate the diffusion phenomenon in a two component system revealed an unstable behavior. When this scheme is used to get the time dependent solutions of the lattice Boltzmann equations (1.1), the probability functions $f_i(x,t)$ become negative very often, forcing the simulation code to be stopped. Although centered finite difference schemes for spatial derivatives arise in a natural way and thus, are easy to introduce, these schemes are known to be unconditionally unstable in accordance to the von Neumann stability analysis [2, 3, 4] applied to hyperbolic equations like the Lattice Boltzmann equations (1.1).

This chapter reports our attempts to find appropriate finite difference schemes for the Lattice Boltzmann equations. These schemes are introduced for the nine bit model using the characteristics curves of hyperbolic equations in one space dimension [4] and we show that all these schemes are equivalent to the former Lattice Gas Lattice Boltzmann (LGLB) scheme [1] when the Courant - Friedrichs - Lewy number CFL (to be defined further) equals unity. All these schemes are based on Lagrange interpolation procedures. Some considerations relative to the treatment of boundary lattice nodes are also introduced here.

1.2 Spatial derivatives on a square lattice

To solve Eqs (1.1) on a square lattice, we need a procedure to compute the following term at the point $\mathbf{r} = (x, y)$

$$\mathbf{e}_{i} \cdot \nabla_{\mathbf{r}} f_{i}(\mathbf{r}, t) = (\mathbf{e}_{i})_{\alpha} \partial_{\alpha} f_{i}(\mathbf{r}, t)$$

$$= (\mathbf{e}_{i})_{x} \partial_{x} f_{i}(x, y, t) + (\mathbf{e}_{i})_{y} \partial_{y} f_{i}(x, y, t)$$

$$(1.2)$$

(implicit summation over cartesian components expressed by Greek indices is always understood).

A possibility is to use the Centered Space (CS) finite difference scheme

$$\partial_x f_i(x, y, t) \simeq \frac{f_i(x + \delta x, y, t) - f_i(x - \delta x, y, t)}{2\delta x}$$
 (1.3)

where δx is the lattice spacing in the x direction. A similar expression holds also for the derivative in the y direction (∂_y) ; the lattice spacings are all

equal $(\delta x = \delta y)$ in the case of the square lattice. As mentioned before, this centered finite difference scheme was found to be inadequate for the simulation of diffusion phenomena in a two component system.

Another possibility is to use the First Order Upwind (FU) finite difference scheme [3]. This scheme takes into account the fact that information propagates along the vector \mathbf{e}_i and thus, one should consider also the sign of its components when writing the corresponding numerical scheme. For the x component, we have

$$\partial_x f_i(x, y, t) \simeq \begin{cases} \frac{f_i(x, y, t) - f_i(x - \delta_x, y, t)}{\delta x} &, & (\mathbf{e}_i)_x > 0\\ \frac{f_i(x + \delta_x, y, t) - f_i(x, y, t)}{\delta x} &, & (\mathbf{e}_i)_x < 0 \end{cases}$$
(1.4)

A Second Order Upwind (SU) finite difference scheme may be also introduced [3]

$$\partial_{x} f_{i}(x, y, t) \simeq \begin{cases} \frac{-3f_{i}(x, y, t) + 4f_{i}(x + \delta_{x}, y, t) - f_{i}(x + 2\delta_{x}, y, t)}{2\delta x} \\ (\mathbf{e}_{i})_{x} > 0 \\ \frac{3f_{i}(x, y, t) - 4f_{i}(x - \delta_{x}, y, t) + f_{i}(x - 2\delta_{x}, y, t)}{2\delta x} \\ (\mathbf{e}_{i})_{x} < 0 \end{cases}$$

$$(1.5)$$

Although the FU scheme was found to be stable, it has the disadvantage of a lattice spacing dependence of the viscosity. This is a general feature of first order schemes (see Section 1.10). In the case of second order schemes (like CS and SU), no lattice spacing dependence of the viscosity is observed, but these schemes are not stable when dealing with sharp interfaces (see next chapter, where the use of these schemes to simulate a diffusion couple is discussed). Other schemes, which are mainly based on the characteristics curve and use an interpolation procedure, are described below.

1.3 Characteristics

We start from the differential form of the LB equations (1.1)

$$\partial_t f_i(\mathbf{r}, t) + \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r}, t) = q_i(\mathbf{r}, t)$$

$$i = 0, 1, \dots N$$
(1.6)

where the source term is

$$q_{i}(\mathbf{r},t) = -\frac{1}{\tau_{p}} \left[f_{i}(\mathbf{r},t) - f_{i}^{eq}(\mathbf{r},t) \right] + \frac{1}{k_{B}T} \mathbf{F}(\mathbf{r},t) \cdot \left[\mathbf{e}_{i} - \mathbf{u}(\mathbf{r},t) \right] f_{i}^{eq}(\mathbf{r},t)$$

$$i = 0, 1, \dots N$$
(1.7)

The homogeneous part of Eqs. (1.6)

$$\frac{D}{Dt} f_i(\mathbf{r}, t) = \partial_t f_i(\mathbf{r}, t) + \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r}, t) = 0$$
 (1.8)

has the solution

$$f_i(\mathbf{r},t) = f_i(\mathbf{r}_0, t_0) = \text{constant}$$
 (1.9)

since the vectors \mathbf{e}_i are all constant. Consequently, for each $i=0,1,\ldots N$, the current point moves across a straight line (the characteristics line) which passes through the node \mathbf{r} of the lattice at the moment t and has the orientation of the vector \mathbf{e}_i . The formal solution of the inhomogeneous evolution equations (1.6) may be written as

$$f_{i}(\mathbf{r}, t + \delta t) = f_{i}(\mathbf{r} - \mathbf{e}_{i}\delta t, t) - \frac{1}{\tau_{p}} \int_{t}^{t + \delta t} \left[f_{i}(\mathbf{r}(t'), t') - f_{i}^{eq}(\mathbf{r}(t'), t') \right] dt'$$

$$+ \frac{1}{k_{B}T} \int_{\delta t}^{t + \delta t} \mathbf{F}(\mathbf{r}(t'), t) \cdot \left[\mathbf{e}_{i} - \mathbf{u}(\mathbf{r}(t'), t') \right] f_{i}^{eq}(\mathbf{r}(t'), t')$$

$$i = 0, 1, \dots N$$

$$(1.10)$$

where the integrals are calculated along the characteristics lines. At the initial moment t'=t, the current point is $\mathbf{r}(t')=\mathbf{r}-\mathbf{e}_i\delta t$ and thus, the

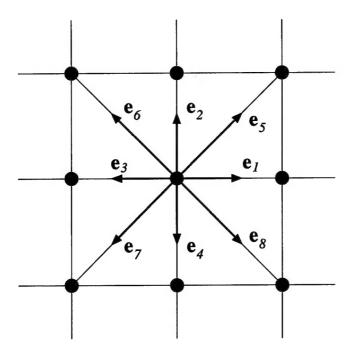


Figure 1.1: Velocity set in the nine bit (N = 8) LB model.

integrals in the equation above may be approximated using the rectangle formula to get

$$f_{i}(\mathbf{r}, t + \delta t) \simeq f_{i}(\mathbf{r} - \mathbf{e}_{i}\delta t, t) - \frac{\delta t}{\tau_{p}} \left[f_{i}(\mathbf{r} - \mathbf{e}_{i}\delta t, t) - f_{i}^{eq}(\mathbf{r} - \mathbf{e}_{i}\delta t, t) \right]$$

$$+ \frac{\delta t}{k_{B}T} \mathbf{F}(\mathbf{r} - \mathbf{e}_{i}\delta t, t) \cdot \left[\mathbf{e}_{i} - \mathbf{u}(\mathbf{r} - \mathbf{e}_{i}\delta t, t) \right] f_{i}^{eq}(\mathbf{r} - \mathbf{e}_{i}\delta t, t)$$

$$i = 0, 1, \dots N$$

$$(1.11)$$

This equation provides the key for the *characteristics* – *based* finite difference schemes to be developed further. Eqs. (1.11) are the expression of the well known Lagrange representation in classical fluid mechanics, where the evolution of a particle on its trajectory is described by the total time derivative.

When using the nine bit LGLB scheme on a square lattice, the characteristics lines always pass through the lattice nodes at any time step. These lines are associated to each velocity speed \mathbf{e}_i in Figure 1.1. For further discussion,

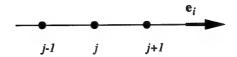


Figure 1.2: Lattice points along the characteristics line.

we will refer to one of these characteristics lines (which corresponds to any of the eight velocities \mathbf{e}_i in Figure 1.1). We may choose a one dimensional reference system whose positive direction is orientated along the speed \mathbf{e}_i and we represent the current points situated along this line as in figure 1.2. In this figure, the current point at the moment t is denoted by j, the next point on the characteristics line (which becomes the current point at the moment $t+\delta t$) is denoted j+1, while the preceding one is denoted j-1. We should remark the fact that the point j (which is the current point on the characteristics line at the moment t) always refers to a lattice node, but the points j-1 and j+1 are superposed to lattice nodes only when using the LGLB scheme, as seen in figure 1.3. This figure shows the current point on the characteristics line, at different time steps, in the K-t plane, when the LGLB scheme is used.

When δx is the spacing between the latice nodess on the reference axis X of the characteristics line (note that δx may be equal to l or to $l\sqrt{2}$, where

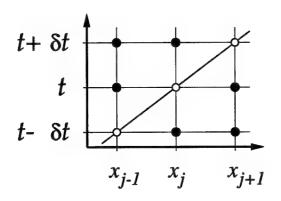


Figure 1.3: Current points (o) on the characteristics line, in the LGLB model.

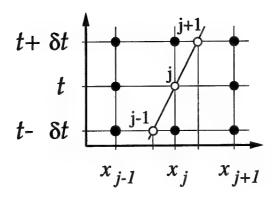


Figure 1.4: Current points (\circ) on the characteristics line, in the Finite Difference LB model (CFL < 1).

l is the side length of the elementary square lattice cell) we have

$$\delta x = x_j - x_{j-1} = x_{j+1} - x_j \tag{1.12}$$

In the LGLB model, where particles are considered to propagate between lattice nodes during one time step, we have

$$\delta x = c\delta t \tag{1.13}$$

where c is the magnitude of the propagation speed c ($c \in \{e_i\}$, i = 1, 2 ... N) and δt is the time step.

When the lattice spacing δx does no more equal the product $c\delta t$, the current nodes j-1 and j+1 on the characteristics line passing through the lattice node x_j (i.e., node j) are no more superposed to lattice nodes. These current points lie now between the lattice node and the neighbors x_{j-1} or x_{j+1} , respectively. This general case is shown in figure 1.4 and is characterized by the Courant - Friedrichs - Lewy number

$$\mathsf{CFL} = \frac{c\delta t}{\delta x} \tag{1.14}$$

When CFL = 1, the former LGLB case is recovered. In accordance to the general theory of hyperbolic equations [2, 3, 4], a necessary (but not sufficient) condition to be satisfied by the finite difference scheme to be stable is

$$\mathsf{CFL} \le 1 \tag{1.15}$$

1.4 Interpolation procedures on the characteristics line

The Lattice Boltzmann equations (1.11) may be solved using an iterative procedure. The value $f_i(\mathbf{r} - \mathbf{e}_i \delta t, t)$ of the distribution function (as well as the value $\mathbf{u}(\mathbf{r} - \mathbf{e}_i \delta t, t)$ of the local velocity) may be calculated using an interpolation procedure along the characteristics line which passes through the lattice node \mathbf{r} at time $t + \delta t$. The current point at the moment t is $\mathbf{r} - \mathbf{e}_i \delta t$, which corresponds to the point x in figure 1.5. If we know the values of the distribution function at at the moment t in two lattice nodes x_1 and x_2 , we may use the first order Lagrange interpolation formula to evaluate the distribution function in the point x

$$f_i(x,t) = \frac{x-x_2}{x_1-x_2} f_i(x_1,t) + \frac{x-x_1}{x_2-x_1} f_i(x_2,t)$$
 (1.16)

The second order Lagrange interpolation formula may be used when three values of the distribution functions are known

$$f_{i}(x,t) = \frac{(x-x_{2})(x-x_{3})}{(x_{1}-x_{2})(x_{1}-x_{3})} f_{i}(x_{1},t) + \frac{(x-x_{1})(x-x_{3})}{(x_{2}-x_{1})(x_{2}-x_{3})} f_{i}(x_{2},t) + \frac{(x-x_{1})(x-x_{2})}{(x_{3}-x_{1})(x_{3}-x_{2})} f_{i}(x_{3},t)$$

$$(1.17)$$

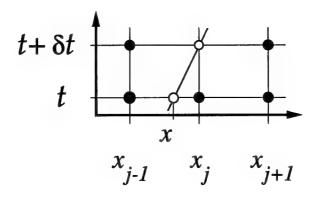


Figure 1.5: Current points (o) on the characteristics line.

Following finite difference schemes may be introduced, depending on the choice of the interpolation formula, as well as on the number and the location of the interpolation points:

- 1. Upwind scheme: the first order interpolation formula (1.16) is used with $x_1 \equiv x_{j-1}$ and $x_2 \equiv x_j$.
- 2. Lax Friedrichs scheme: the first order interpolation formula (1.16) is used with $x_1 \equiv x_{j-1}$ and $x_2 \equiv x_{j+1}$.
- 3. Lax Wendroff scheme: the second order interpolation formula (1.17) is used with $x_1 \equiv x_{j-1}$, $x_2 \equiv x_j$ and $x_3 \equiv x_{j+1}$.

These finite difference schemes, as well as the space centered scheme are discussed below.

1.5 Upwind scheme

The upwind scheme is recovered when using the first order interpolation formula (1.16) with

$$x = x_j - c\delta t$$

$$x_1 = x_{j-1}$$

$$x_2 = x_j$$
(1.18)

In this case, the interpolation formula (1.16) gives

$$f_i(x_i - c\delta t, t) = f_i(x_j, t) - \text{CFL} \left[f_i(x_j, t) - f_i(x_{j-1}, t) \right]$$
 (1.19)

A particular attention should be given to boundary nodes. When the vector \mathbf{e}_i is pointing outwards the lattice domain, the interpolation formula (1.19) may be used. But the interpolation formula should be somewhat different when th vector \mathbf{e}_i points into the lattice domain: if x_j is a node located on the boundary, there is no corresponding x_{j-1} node in this case. The node x_{j+1} may be used instead, and we have the following extrapolation formula in this case:

$$f_i(x_j - c\delta t, t) = f_i(x_j, t) - CFL [f_i(x_{j+1}, t) - f_i(x_j, t)]$$
 (1.20)

After interpolating all distribution functions f_i , i = 0, 1 ... N in the current point $\mathbf{r} - \mathbf{e}_i \delta t$, the local velocity $\mathbf{u}(\mathbf{r} - \mathbf{e}_i \delta t, t)$ should be evaluated and introduced in Eq. (1.11) which allows to calculate the new values of the distribution functions in all lattice nodes.

Since the values of the velocity components are prescribed on the boundaries, an interpolation procedure becomes necessary to evaluate also the velocity components at the current point $\mathbf{r} - \mathbf{e}_i \delta t$ when the node \mathbf{r} is located on the boundary. Interpolation and extrapolation formulae similar to Eqs. (1.19) and (1.20) give the value of the velocity $\mathbf{u}(x_j - c\delta t, t)$, which allows the calculation of the equilibrium distribution function $f_i^{eq}(x_j - c\delta t, t)$ in Eq. (1.11).

A very important feature of the upwind finite difference scheme (1.19) is the recovery of the former Lattice Gas like Latice Boltzmann (LGLB) model when CFL = 1. The upwind scheme says that the information (i.e., the distribution function f_i) available at the moment t at the current point $x_j - c\delta t$ is processed there as a result of collisions (through the relaxation term mainly characterized by the physical relaxation time τ_p , as well as the action of some force F), and thereafter arrives in node x_j at the moment $t + \delta t$. But, when CFL = 1, the initial current point $x_j - c\delta t$ is identical to the lattice node x_{j-1} , in accordance to Eq. (1.19). This feature belongs also to the Lax-Friedrichs and the Lax - Wendroff schemes to be discussed below. In this respect, all these characteristics curve derived finite difference schemes provide a natural generalisation of the former LGLB model and allow to separate the lattice spacing δx from the particles free path $c\delta t$. This separation process provides the keystone to the LB simulation of multicomponent particle systems.

1.6 Lax - Friedrichs scheme

The Lax - Friedrichs scheme [2] is recovered when using the first order interpolation formula (1.16) with the following interpolation points

$$x = x_j - c\delta t$$

$$x_1 = x_{j-1}$$

$$x_2 = x_{j+1}$$

$$(1.21)$$

The interpolation formula (1.16) gives, at the moment t

$$f_i(x_j - c\delta t, t) = \frac{f_i(x_{j+1}, t) + f_i(x_{j-1}, t)}{2}$$

- CFL
$$\frac{f_i(x_{j+1},t) - f_i(x_{j-1},t)}{2}$$
 (1.22)

At the boundary nodes, the Lax - Friedrichs scheme cannot be used, but the upwind scheme may be accepted instead.

1.7 Space centered scheme

If we approximate

$$\frac{f_i(x_{j+1},t) + f_i(x_{j-1},t)}{2} \approx f_i(x_j,t)$$
 (1.23)

in the Lax Friedrichs scheme (1.22), we get the usual space centered scheme

$$f_i(x_j - c\delta t, t) = f_i(x_j, t) - CFL \frac{f_i(x_{j+1}, t) - f_i(x_{j-1}, t)}{2}$$
 (1.24)

This scheme is known the be unconditionally unstable [2, 3]. We really get negative values of the distribution functions when using this scheme, after a certain number of time steps, and this situation forces the LB code to be stopped. The number of time steps before the space centered computer code crashes is dependent on the CFL number, being larger when this number becomes smaller. Using a smaller Courant - Friedrichs - Lewy number for a given lattice spacing δx is unconvenient since this means a smaller time step δt , which generates a huge increase of the number of necessary time steps to be performed in order to recover the system evolution during a certain time interval.

1.8 Lax - Wendroff scheme

The second order Lagrange interpolation formula (1.17) and the following interpolation points

$$x = x_j - c\delta t$$

$$x_1 = x_{j-1}$$

$$x_2 = x_j$$

$$x_3 = x_{j+1}$$
(1.25)

give the Lax - Wendroff finite difference scheme

$$f_{i}(x_{j} - c\delta t, t) = \frac{\mathsf{CFL}(1 + \mathsf{CFL})}{2} f(x_{j-1}, t)$$

$$- (\mathsf{CFL} - 1)(\mathsf{CFL} + 1) f_{i}(x_{j}, t)$$

$$- \frac{\mathsf{CFL}(1 - \mathsf{CFL})}{2} f(x_{j+1}, t) \tag{1.26}$$

As for the Lax - Friedrichs scheme, the Lax - Wendroff scheme (1.26) cannot be applied in the boundary nodes. One possibility is to use there the upwind schemes (1.19) and (1.20) for outgoing and ingoing speed vectors \mathbf{e}_i , respectively. Another possibility is to preserve the second order interpolation degree and to use the following expressions

$$f_{i}(x_{j} - c\delta t, t) = \frac{\mathsf{CFL}(\mathsf{CFL} - 1)}{2} f_{i}(x_{j-2}, t)$$

$$+ \mathsf{CFL}(2 - \mathsf{CFL}) f_{i}(x_{j-1}, t) \qquad (1.27)$$

$$+ \frac{(1 - \mathsf{CFL})(2 - \mathsf{CFL})}{2} f_{i}(x_{j}, t)$$

$$f_{i}(x_{j} - c\delta t, t) = \frac{(1 + \mathsf{CFL})(2 + \mathsf{CFL})}{2} f_{i}(x_{j}, t)$$

$$- \mathsf{CFL}(2 + \mathsf{CFL}) f_{i}(x_{j+1}, t)$$

$$- \frac{\mathsf{CFL}(1 + \mathsf{CFL})}{2} f_{i}(x_{j+2}, t) \qquad (1.28)$$

for the outgoing and ingoing speed vectors \mathbf{e}_i , respectively. Note that, in these expressions, the node x_j is always on the lattice border, while the positive direction (from node x_j to node x_{j+1} is the same as the direction of the speed vector \mathbf{e}_i .

1.9 Interpolation Supplemented Schemes

1.9.1 General Description

One can see that the procedure described by Eq. (1.11) implies the following computation steps:

- 1. Lagrange interpolation to find the distribution functions, as well as the local velocities in the points $\mathbf{r} \mathbf{e}_i \delta t$; these points are not lattice nodes and are all different for $i = 1, 2, \ldots N$
- 2. collisions processing and the addition of the effect of external forces
- 3. propagation from the point $\mathbf{r} \mathbf{e}_i \delta t$ to the lattice node \mathbf{r}

For each lattice node \mathbf{r} , one has to perform $N \times (N+1)$ interpolations since there are N directions \mathbf{e}_i and N+1 distribution functions to be interpolated. The function f_0 should also be interpolated since its value in the point $\mathbf{r} - \mathbf{e}_i \delta t$ is needed to calculate the value of the local speed $\mathbf{u}(\mathbf{r} - \mathbf{e}_i \delta t, t)$; the local speed $\mathbf{u}(\mathbf{r} - \mathbf{e}_i \delta t, t)$ is necessary to calculate the equilibrium distribution function $f_i(\mathbf{r} - \mathbf{e}_i \delta t, t)$.

An equivalent procedure was already proposed in the LB litterature [5, 6, 7, 8]. This *Interpolation Supplemented Lattice Boltzmann* (ISLB) scheme (as named by their authors) has the same steps as described above, but their order is changed:

- 1. the collisions and the action of external forces are processed in the lattice nodes ${f r}$
- 2. the propagation step transports the result towards the points $\mathbf{r} + \mathbf{e}_i \delta t$
- 3. the interpolation step recovers the new distribution functions in the lattice node \mathbf{r} at time $t + \delta t$

To avoid confusion with another scheme to be described below, we will denote this original ISLB scheme as ISLB-CP (ISLB Collision – Propagation) since the collision step is done before the propagation step. Figure 1.6 recalls the main characteristics of this ISLB-CP scheme. In this figure, lattice nodes are denoted by $0, 1, \ldots 8$. Let $\mathbf{r}_0, \mathbf{r}_1, \ldots \mathbf{r}_8$ the position vectors of these lattice nodes. After processing the effect of collisions (as well as the effect

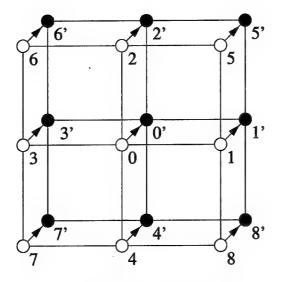


Figure 1.6: The ISLB-CP scheme.

of external forces) in the original lattice nodes, particle distribution functions are propagated along their characteristics lines during the time step δt and arrive in the displaced nodes $0', 1', \dots 8'$, whose position vectors are $\mathbf{r}'_0, \mathbf{r}'_1, \dots \mathbf{r}'_8$ (figure 1.6 shows the case when the displacement is made along the \mathbf{e}_5 direction). Thus, for each vector $\mathbf{e}_i, i = 1, 2, \dots 8$, the collision procedure, followed by the propagation procedure, transforms the initial set of distribution functions $\{f_i(\mathbf{r}_n, t)\}, n = 0, 1 \dots 8$ into the set $\{f_i(\mathbf{r}'_n, t + \delta t)\}, n' = 0', 1', \dots 8'$. A 2D interpolation procedure (to be described further) allows to recover the value of $f_i(\mathbf{r}_0, t + \delta t)$ [5, 6, 7] and thus, the automaton algorithm may be applied again for the next time step, and so on.

It is possible to imagine an ISLB scheme where propagation is done first, followed by collisions processing. This scheme is presented in figure 1.7. If we start from the Lattice Boltzmann Equations (1.11) we may write

$$\mathbf{x} = \mathbf{r} - \mathbf{e}_i \delta t$$

$$f_i(\mathbf{x}, t) = f_i(\mathbf{r}, t) - \delta t \, \mathbf{e}_i \cdot \nabla_{\mathbf{r}} f_i(\mathbf{r}, t) \qquad (1.29)$$

and thus

$$f_i(\mathbf{r},t+\delta t) \simeq f_i(\mathbf{x},t) - \frac{\delta t}{\tau_p} \left[f_i(\mathbf{x},t) - f_i^{eq}(\mathbf{x},t) \right] +$$

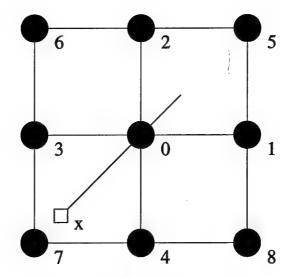


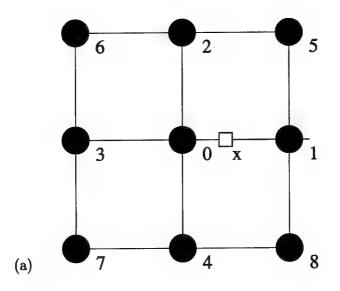
Figure 1.7: The ISLB-PC scheme.

$$\frac{\delta t}{k_B T} \mathbf{F}(\mathbf{x}, t) \cdot [\mathbf{e}_i - \mathbf{u}(\mathbf{x}, t)] f_i^{eq}(\mathbf{x}, t)$$

$$i = 0, 1, \dots N$$
(1.30)

In this scheme, which we will denote as ISLB-PC (since the Propagation step is done before the Collision step), an interpolation procedure in two dimensions is used to determine the value of $f_i(\mathbf{x},t)$ in Eq. (1.30). To find the value of $f_i(\mathbf{x},t)$ when one has the values $\{f_i(\mathbf{r}_n,t)\}$, $n=0,1\ldots 8$, we may use the same interpolation procedure as for the ISLB-CP scheme, which gives $f_i(\mathbf{r}_0,t+\delta t)$ from the set $\{f_i(\mathbf{r}'_n,t+\delta t)\}$, $n'=0',1',\ldots 8'$. The ISLB-PC approach may be viewed as being equivalent to the early Finite Difference schemes developed in [1], but replaces the calculation of local derivatives through an interpolation procedure, while maintaining the propagation of information along the characteristics line.

The ISLB-CP and ISLB-PC schemes use biquadratic interpolation to calculate the values of the distribution functions in the central node (denoted by 0 in Figures 1.6 and 1.7). This is not the unique possibility. In fact, the authors who introduced the original ISLB scheme [5, 6, 7, 8] reccommmended the use of an upwind biquadratic interpolation procedure. This procedure is explained briefly in Figure 1.8, where the position of the point \mathbf{x} is shown



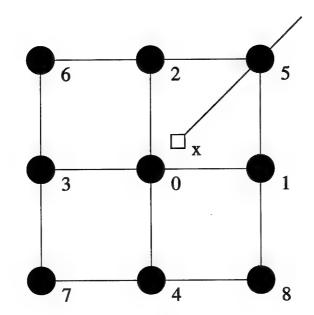


Figure 1.8: The UISLB-PC scheme.

(b)

in two cases, when the characteristics lines are directed along the velocity vectors \mathbf{e}_5 and \mathbf{e}_1 , respectively (the other velocity directions may be considered in a similar way). In these Upwind ISLB-PC (UISLB-PC) cases, the distribution functions are interpolated in node x and thereafter propagated on the corresponding characteristics lines towards the lattice nodes 5 (Figure 1.8a) and 1 (Figure 1.8b). An Upwind ISLB-CP (UISLB-CP) scheme may be introduced in the same way as the UISLB-PC scheme described here.

The Iterpolation Supplemented Lattice Boltzmann schemes were originally developed to deal with non - uniform grids. To our best knowledge, in the LB litterature there is no attempt to use ISLB schemes to multiple component systems whose particles carry different masses. When developing the ISLB-PC scheme, their authors started from the early LGLB philosophy, which considers that distribution functions are redistributed at each lattice node because of collisions and are moved thereafter in the direction of their corresponding velocity vectors. When compared to this philosophy, the present discussion of Finite Difference Schemes for the Boltzmann Equation, which is based on the characteristics lines of the Boltzmann equation, may be another step towards a more rigorous approach to Lattice Boltzmann models. Because the Lattice Boltzmann equation is a hyperbolic equation, the characteristics curves (which are straight lines since the velocity vectors \mathbf{e}_i are constant [4]) may provide the background for error analysis or stability investigations.

1.9.2 Interpolation procedures on the 2D lattice

Both ISLB-CP and ISLB-PC schemes need an interpolation procedure to evaluate the value of the distribution function $f_i(\mathbf{x},t)$ at the point $\mathbf{x}=(x,y)=\mathbf{r}_0-\mathbf{e}_i\delta t$ in figure 1.7, when the corresponding values $\{f_i(\mathbf{r}_n,t)\}$, $n=0,1,\ldots 8$ of the distribution function are known. As pointed in [8], second order interpolation is necessary in order to avoid spurious dependence of the fluid viscosity on the lattice size.

If we introduce a cartesian coordinate system centered in the node $\mathbf{r}_0 = (x_0, y_0)$ in figure 1.7, such that the X axis points along the vector \mathbf{e}_1 and we use the fact that we deal with a square lattice, we may define the non-dimensional (natural) coordinates [9]

$$\xi = \frac{x - x_0}{\delta x} \in [-1, +1]$$

$$\eta = \frac{y - y_0}{\delta x} \in [-1, +1] \tag{1.31}$$

To simplify the notation, in this subsection we will write $f(\mathbf{x})$ instead of $f_i(\mathbf{x},t)$, and $f(\mathbf{r}_n)$, $n=0,1,\ldots 8$ instead of $f_i(\mathbf{r}_n,t)$. The interpolation procedure gives

$$f(\mathbf{x}) = \sum_{n=0}^{n=8} N_n f(\mathbf{r}_n)$$
 (1.32)

where the weight coefficients N_n are given in table 1.1 as the product of two second order Lagrange interpolation coefficients (one for each axis), which are defined as follows (the argument θ stands for ξ or η)

$$L_{-}^{2}(\theta) = \frac{[\theta - 0][\theta - 1]}{[(-1) - 0][(-1) - 1]}$$

$$L_{0}^{2}(\theta) = \frac{[\theta - (-1)][\theta - 1]}{[0 - (-1)][0 - 1]}$$

$$L_{+}^{2}(\theta) = \frac{[\theta - (-1)][\theta - 1]}{[1 - (-1)][1 - 0]}$$
(1.33)

The biquadratic interpolation procedure described above may be applied to any lattice node which is not situated on the boundary (i.e., bulk nodes). Bulk nodes \mathbf{r}_0 have all the eight neighbors \mathbf{r}_n , $n=0,1,\ldots 8$. When the node \mathbf{r}_0 is located on the lattice boundary, some of these neighbors are missing, e.g., as shown in figure 1.9, which refers to four kinds of boundaries (walls): bottom (B), top (T), left (L) and right (R), Such situations are encountered when one deals with flow between horizontal or parallel plates (we will not discuss here the case when the fluid system is confined into a box – this case needs to consider also the presence of corner nodes).

For wall nodes, Eq. (1.32) changes to

$$f(\mathbf{x}) = \sum_{n \in \mathcal{N}_W} N_n f(\mathbf{x}_n)$$
 (1.34)

where the sum is computed with respect to all indices in the set \mathcal{N}_W , $W \in \{B, T, L, R\}$. The elements of the sets \mathcal{N}_B are (see figure 1.9):

$$\mathcal{N}_B \equiv \{0, 1, 2, 3, 5, 6\}$$

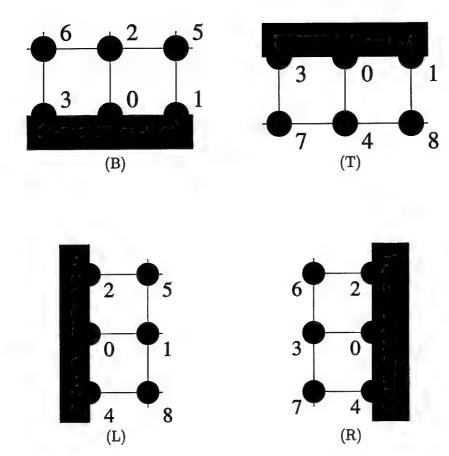


Figure 1.9: Lattice nodes on the walls: (B) – bottom wall; (T) – top wall; (L) – left wall; (R) – right wall.

Table 1.1: Interpolation weights for the bulk lattice nodes.

N_0	$L^2_0(\xi)L^2_0(\eta)$
N_1	$L^2_+(\xi)L^2_0(\eta)$
N_2	$L_0^2(\xi)L_+^2(\eta)$
N_3	$L^2(\xi)L^2_0(\eta)$
N_4	$L_0^2(\xi)L^2(\eta)$
N_5	$L^2_+(\xi)L^2_+(\eta)$
N_6	$L^2(\xi)L^2_+(\eta)$
N_7	$L^2(\xi)L^2(\eta)$
N_8	$L^2_+(\xi)L^2(\eta)$

$$\mathcal{N}_{T} \equiv \{0, 1, 3, 4, 7, 8\}$$
 $\mathcal{N}_{L} \equiv \{0, 1, 2, 4, 5, 8\}$
 $\mathcal{N}_{R} \equiv \{0, 2, 3, 4, 6, 7\}$

We may use linear Lagrange interpolation for the Y directions when dealing with horizontal walls, or for the X direction when dealing with vertical walls. To this purpose, we define the corresponding coefficients

$$L_0^{1-}(\theta) = \frac{\theta - (-1)}{0 - (-1)}$$

$$L_-^{1-}(\theta) = \frac{\theta - 0}{(-1) - 0}$$

$$L_0^{1+}(\theta) = \frac{\theta - 1}{0 - 1}$$

$$L_+^{1-}(\theta) = \frac{\theta - 0}{1 - 0}$$
(1.35)

while the interpolation weights N_n are given in table 1.2, for each boundary.

Table 1.2: Interpolation weights for the boundary lattice nodes.

bottom	N_0	$L^2_0(\xi)L^{1+}_0(\eta)$
	N_1	$L^2_+(\xi)L^{1+}_0(\eta)$
	N_2	$L^2_0(\xi)L^{1+}_+(\eta)$
	N_3	$L^2(\xi)L^{1+}_0(\eta)$
	N_5	$L^2_+(\xi) L^{1+}_+(\eta)$
	N_6	$L^2(\xi) L^{1+}_+(\eta)$
top	N_0	$L^2_0(\xi)L^{1-}_0(\eta)$
	N_1	$L^2_+(\xi)L^{1-}_0(\eta)$
	N_3	$L^2(\xi) L^{1-}_0(\eta)$
	N_4	$L^2_+(\xi) L^{1-}(\eta)$
	N_7	$L^{2}_{-}(\xi) L^{1-}_{-}(\eta)$
	N_8	$L_0^2(\xi) L^{1-}(\eta)$
left	N_0	$L_0^0(\xi)L_0^2(\eta)$
	N_1	$L^{1+}_+(\xi)L^2_0(\eta)$
	N_2	$L_0^{1+}(\xi) L_+^2(\eta)$
	N_4	$L_0^{1+}(\xi) L^2(\eta)$
	N_5	$L_{+}^{1+}(\xi) L_{+}^{2}(\eta)$
	N_8	$L_{+}^{1+}(\xi) L_{-}^{2}(\eta)$
right	N_0	$L_0^{1-}(\xi)L_0^2(\eta)$
•	N_2	$L_0^{1-}(\xi) L_+^2(\eta)$
	N_3	$L^{1-}_{-}(\xi) L^{2}_{0}(\eta)$
	N_4	$L_0^{1-}(\xi)L^2(\eta)$
	N_6	$L^{1-}_{-}(\xi) L^{2}_{+}(\eta)$
	N_7	$L^{1-}_{-}(\xi) L^{2}_{-}(\eta)$

We should underline the fact that the ISLB schemes (ISLB-CP, ISLB-PC, UISLB-CP and UISLB-PC), which use second order (biquadratic) 2D interpolation procedures, are different from the previously introduced Upwind, Lax - Friedrichs, Space Centered and Lax - Wendroff schemes, which use 1D interpolation. We want to point that other 2D interpolation schemes may be also considered, besides the ISLB schemes (i.e., biquadratic interpolation procedure adopted here for bulk nodes, associated with a linear - quadratic procedure for boundary nodes). For example, the bilinear Lagrange interpolation, as well as the linear or the quadratic elements in the "serendipity family" [9] may provide some alternatives.

1.10 Numerical simulations

1.10.1 Computer code

In the remaining section of this chapter, we are reporting a few significant simulation results which were obtained using the explicit Finite Difference Lattice Boltzmann (FDLB) schemes introduced above. These results refer to 2D Poisseuille flow and were mainly done to see the behavior and to estimate the errors and performances of each scheme we investigated.

To reduce the programming effort, we restricted ourserves to the nine bit model and used a square lattice. The computer code is given in Appendix A. All the necessary simulation parameters (number of lattice nodes in the X and Y direction, number of cycles to be performed, numerical scheme to be used, boundaries, initial configuration, particle masses, local number densities, wall velocities and so on) are introduced in the input data file wet9.input. The input parameter key_scheme controls the numerical scheme to be used during the simulation, as shown in Table 1.3. For further details concerning the meaning of input parameters in the file wet9.input please refer to the comment lines in the source code.

The other chapters of our present report will be dedicated to the physics of diffusion, surface tension and wetting phenomena, as it can be recovered using the computer codes based on the explicit numerical schemes we developed.

1.10.2 Single Component Fluid

We made different computer runs to study the influence of the parameters δx , δt and τ_p on the kinematic viscosity ν . Initially, the flow of a single component fluid within a 2D channel (Poisseuile flow) was simulated using the Upwind, Lax - Friedrichs and Lax - Wendroff schemes. A 5×25 lattice was used, with the same forcing term. The value of the viscosity was determined after fitting the parabolic velocity profile across the channel, as done in our intermediate report [1].

Figure 1.10 shows the influence of the lattice spacing δx on the kinematic viscosity ν , when the other parameters are constant ($\tau_p = 10^{-8}$ s, $\delta t = 10^{-9}$ s, particle mass m=1 amu, temperature T=300 K, force $F=10^{-22}$ N). When using the first order schemes (Upwind, Lax - Friedrichs or FU), the viscosity is always found to increase when the lattice spacing δx increases. In particular, when using the Upwind or the FU scheme, a linear dependence of

Table 1.3: Possible values of the input parameter key_scheme, which controls the numerical scheme to be used during computer simulations.

key_scheme	Numerical Scheme	Section			
0	Upwind (U)	1.5			
1	Lax – Friedrichs (LF)	1.6			
2	Space Centered (SC)	1.7			
3	Lax – Wendroff (LW)	1.8			
7	ISLB – CP	1.9			
8	UISLB - CP	1.9			
9	ISLB - PC	1.9			
10	UISLB - PC	1.9			
11	Linear Serendip Elements	1.9			
12	Bilinear Interpolation	1.9			
13	First Order Upwind (FU)	1.2			
14	Centered Space (CS)	1.2			
15	Second Order Upwind (SU)	1.2			

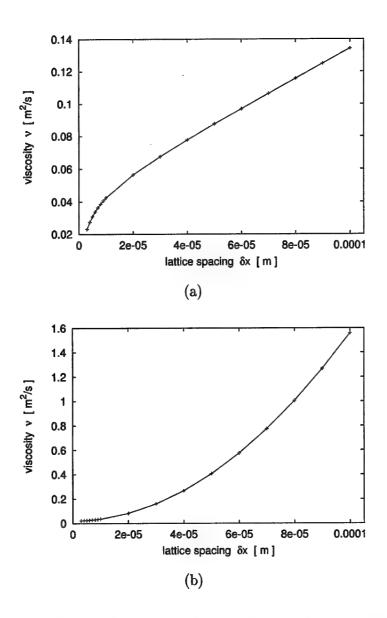


Figure 1.10: Influence of the lattice spacing δx on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

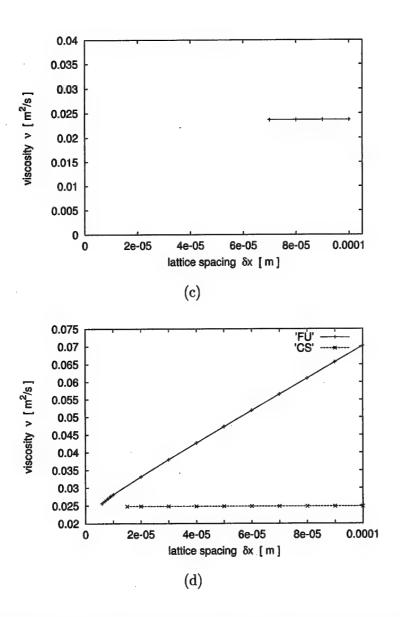


Figure 1.10: (continued) Influence of the lattice spacing δx on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

the viscosity on the lattice spacing δx is found when this parameter becomes large enough. The kinematic viscosity ν is found to be independent on the lattice spacing only when using second order schemes (Lax - Wendroff or CS). Similar results were reported when using the ISLB scheme [5, 6]. A rigorous analysis of an 1D ISLB model [8], revealed that one should use a second order interpolation formula in order to avoid a spurious (lattice spacing dependent) viscosity term in the Navier Stokes equation.

Despite the recovery of the correct (lattice spacing independent) value of the viscosity, the Lax Wendroff scheme in our computer code is found to become unstable when the Courant - Friedrichs - Lewy number exceeds a certain value ($CFL \simeq 0.04$). This is why the values $\nu = \nu(\delta x)$ in figure 1.10c are available only for $\delta x \geq 7 \times 10^{-5}$ m. This behavior may be associated to the necessity to keep a small value of the Courant - Friedrichs - Lewy number CFL, in order to preserve the validity of the first order series expansion (1.29), which is essential to all interpolation based FDLB scheme. A similar problem was observed when using the CS scheme (Figure 1.10d) but, in this case, the lattice spacing δx should be larger than 1.5×10^{-5} m (which means $CFL \simeq 0.2$) in order to avoid the negative values of the equilibrium distribution functions.

Figure 1.11 shows the influence of the time step δt on the kinematic viscosity, when $\delta x = 10^{-4}$ m and the other parameters were unchanged. When using the Upwind, Lax - Friedrichs or Lax - Wendroff schemes, the viscosity ν is always found to decrease when increasing the time step. Only in the case of the Lax - Wendroff scheme, the dependence $\nu = \nu(\delta t)$ is a linear one, in accordance to the general theory of the LGLB model [1, 10, 11]

$$\nu = \frac{2\tau_p - \delta t}{2} \chi c^2 = \frac{2\tau_p - \delta t}{2} \frac{k_B T}{m}$$
 (1.36)

The viscosity is found to be independent on the time step when using the Finite Difference Schemes (FU and CS). This is in accordance to the viscosity formula derived previously in our intermediate report [1]

$$\nu = \tau_p \chi c^2 = \frac{\tau_p}{m} k_B T \tag{1.37}$$

Figure 1.12 shows the influence of the relaxation time τ_p , when $\delta x = 10^{-4}$ m and $\delta t = 10^{-9}$ s. Eq. (1.36), which predicts a linear dependence of the viscosity on the relaxation time τ_p , is found again to be valid for the Lax - Wendroff scheme, while Eq. (1.37) is well verified in the case of the Centered

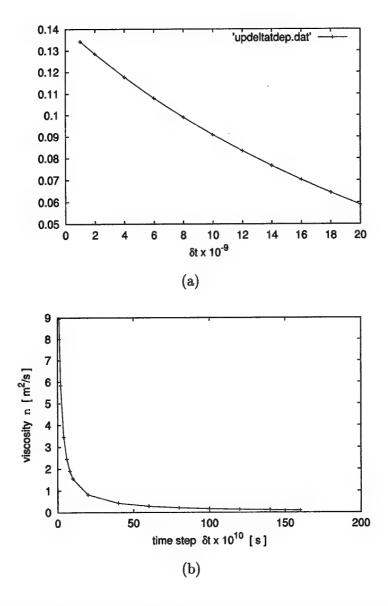


Figure 1.11: Influence of the time step δt on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

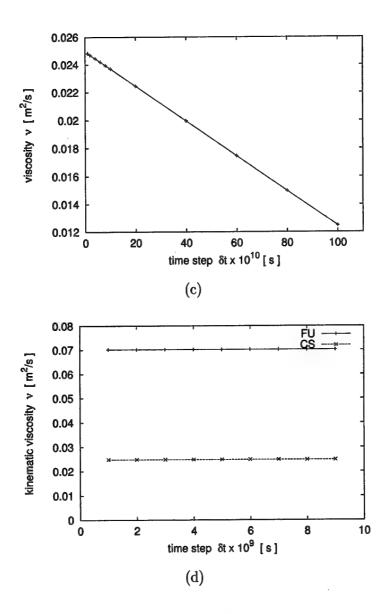


Figure 1.11: (continued) Influence of the time step δt on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

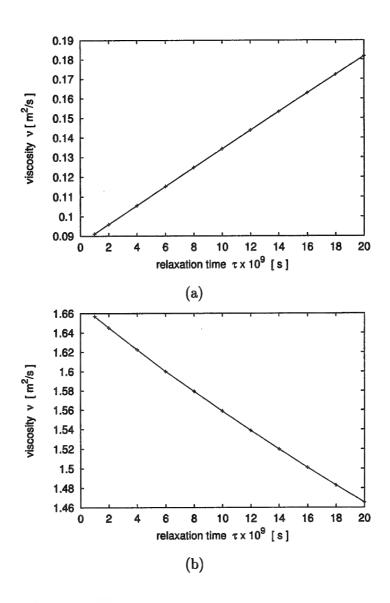


Figure 1.12: Influence of the relaxation time τ_p on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme.

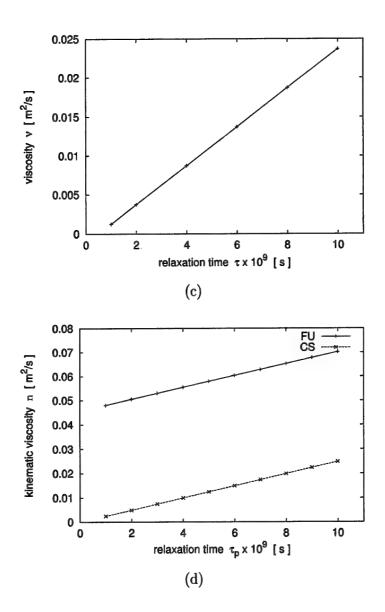


Figure 1.12: (continued) Influence of the relaxation time τ_p on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme; (c) - Lax - Wendroff scheme; (d) - FU and CS schemes.

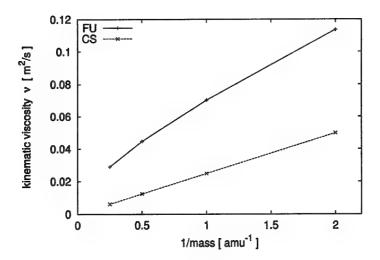


Figure 1.13: Dependence of the kinematic viscosity ν on the inverse of the mass (m^{-1}) of the particles in the fluid system, as determined by fitting the parabolic velocity profile in a 2D channel (FU and CS schemes).

Space (CS) scheme. Even if the Upwind and the First Order Upwind (FU) schemes give also a linear dependence $\nu = \nu(\tau_p)$, the values of the viscosity are not within the range predicted by Eqs. (1.36) and (1.37), respectively.

Although the Lax - Wendroff scheme gives the correct dependence of the viscosity on the simulation parameters δx , δt and τ_p , we found it to be unstable when dealing with a homogeneous two component fluid system. It is this reason why we turned later to other schemes, which proved good stability for such systems.

To check the FU and CS schemes further, we performed several runs with $\delta x = 0.0001$ m, $\delta t = 10^{-9}$ s, $\tau_p = 10^{-8}$ s and different values of the mass m of the particles. The results are shown in Figure 1.13. A linear dependence of the kinematic viscosity on the inverse of the mass (m^{-1}) , which is predicted by Eq. (1.37) is found only in the case of the CS scheme.

Table 1.4 and Figure (1.14) show the dependence of the kinematic viscosity ν on the lattice spacing δx when centered ISLB schemes are used. Two values of the time step δt were adopted, while the relaxation time was maintained constant ($\tau_p = 1 \times 10^{-8}$ s). One may see that table 1.4 has a few missing entries in the column $\delta t = 2 \times 10^{-9}$ s. These missing en-

Table 1.4: Influence of the lattice spacing δx on the viscosity (ISLB schemes).

	δx	u	u
	[s]	$[m^2/s]$	$[m^2/s]$
21		$\delta t = 1 \times 10^{-9} \text{ s}$	$\delta t = 2 \times 10^{-9} \text{ s}$
ISLB-CP	0.3×10^{-5}	0.020865	
	$0.5 imes 10^{-5}$	0.022264	
	$0.6 imes 10^{-5}$	0.022575	0.021173
	0.7×10^{-5}	0.022785	0.021425
	$1.0 imes 10^{-5}$	0.023132	0.021829
	$2.0 imes 10^{-5}$	0.023457	0.022198
	$3.0 imes 10^{-5}$	0.023532	0.022284
	$4.0 imes 10^{-5}$	0.023560	0.022317
	$5.0 imes 10^{-5}$	0.023573	0.022332
	10.0×10^{-5}	0.023592	0.022354
ISLB-PC	0.3×10^{-5}	0.020059	
	0.5×10^{-5}	0.021926	
	0.6×10^{-5}	0.022332	0.020750
	0.7×10^{-5}	0.022603	0.021105
	1.0×10^{-5}	0.023093	0.021665
	$2.0 imes 10^{-5}$	0.023433	0.022155
	3.0×10^{-5}	0.023521	0.022265
	$4.0 imes 10^{-5}$	0.023554	0.022306
	5.0×10^{-5}	0.023577	0.022326
	10.0×10^{-5}	0.023592	0.022352

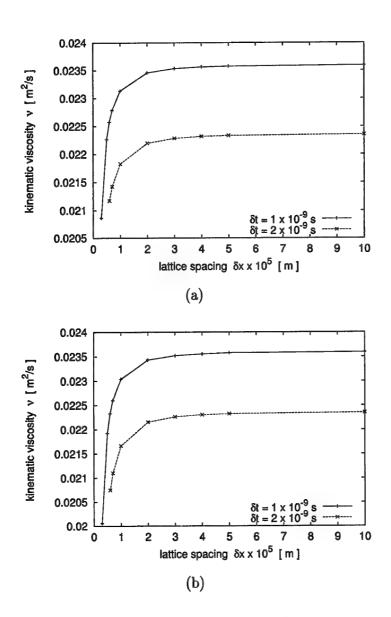


Figure 1.14: Influence of the lattice spacing δx on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel, for two values of the time step δt (1 × 10⁻⁹ s and 2 × 10⁻⁹: (a) - ISLB-CP scheme; (b) - ISLB-PC scheme.

tries correspond to CFL > 1, when the numerical scheme is always unstable and distribution functions become negative. For both $\delta t=1\times 10^{-9}$ s and $\delta t=2\times 10^{-9}$ s, the kinematic viscosity increases when increasing the lattice spacing δx (i.e., when decreasing the Courant - Friedrichs - Lewy number CFL below unity). When the lattice spacing is large enough to achieve CFL < 0.1 (approximatively), the kinematic viscosity becomes very close to the theoretical value calculated in accordance to Eq. (1.36). This feature is seen clearly in Figure 1.14, where the curves $\nu=\nu(\delta x)$ exhibit a saturation behavior. Thus, the error in the determination of the viscosity becomes negligible (less that one percent in our case) when the first order series expansion (1.29) becomes accurate enough.

In order to see the influence of the time step δt on the viscosity, we performed several computer runs with the ISLB schemes, using a fixed value of the lattice spacing δx . This value was choosen to be $\delta x = 3.0 \times 10^{-5}$ m, which lies within the interval where the approximation (1.29) is acceptable. The results, which are reproduced in Table 1.5, as well as in Figure (1.15, show a linear dependence $\nu = \nu(\delta t)$, as expected in accordance to Eq. (evisco). This equation predicts also a linear dependence of the kinematic viscosity ν with respect to the relaxation time τ_p , which is seen in Figure 1.16, for $\delta x = 3.0 \times 10^{-5}$ m and $\delta_t = 1.0 \times 10^{-9}$ s

The numerical experiments described in this subsection revealed that only the CS scheme, as well as the ISLB-CP and ISLB-PC schemes give the correct dependence of the fluid viscosity on the simulation parameters, for a single component fluid.

Table 1.5: Influence of the time step δt on the kinematic viscosity ν (ISLB schemes).

	δt [s]	$ \frac{\nu}{\mathrm{[m^2/s]}} $
ISLB-CP	0.5×10^{-9} 1.0×10^{-9} 1.5×10^{-9} 2.0×10^{-9} 2.5×10^{-9} 3.0×10^{-9}	0.024165 0.235323 0.022906 0.022284 0.021665 0.021047
ISLB-PC	0.5×10^{-9} 1.0×10^{-9} 1.5×10^{-9} 2.0×10^{-9} 2.5×10^{-9} 3.0×10^{-9}	0.024159 0.023521 0.022891 0.022265 0.021642 0.021022

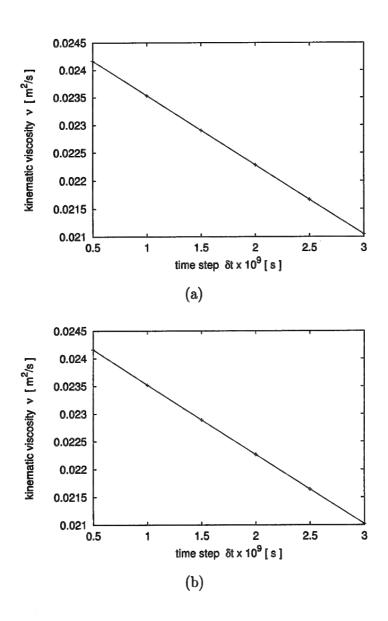


Figure 1.15: Influence of the time step δt on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel: (a) - ISLB-CP scheme; (b) - ISLB-PC scheme.

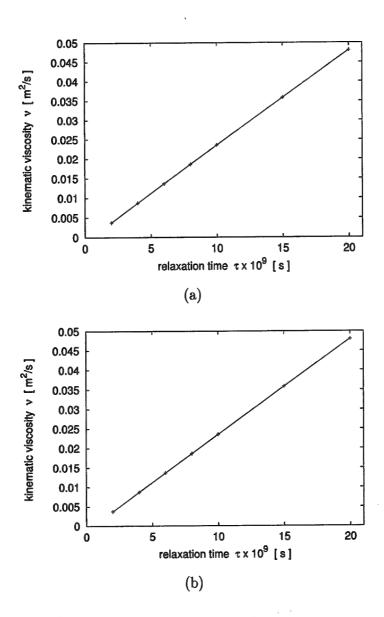


Figure 1.16: Influence of the relaxation time τ_p on the kinematic viscosity ν , as determined by fitting the parabolic velocity profile in a 2D channel, for a fixed of the time step δt (1 × 10⁻⁹ s): (a) - ISLB-CP scheme; (b) - ISLB-PC scheme.

1.10.3 Homogeneous Two Component Fluid

We consider a homogeneous two component fluid system subjected to Poisseuille flow, whose composition may be varied. The system's composition may be described using the mass composition

$$\omega = \frac{\rho_1}{\rho_1 + \rho_2} = \frac{\rho_1}{\rho} \tag{1.38}$$

where ρ_1 and ρ_2 are the mass densities of the two components and $\rho = \rho_1 + \rho_2$ is the mass density of the whole system. Another possibility to account for the composition of the homogeneous system is to use the mole fraction

$$x = \frac{n_1}{n_1 + n_2} = \frac{n_1}{n} \tag{1.39}$$

where n_1 and n_2 are the number of moles belonging to each component and $n = n_1 + n_2$ is the total number of moles in the system.

In order to study the composition dependence of the system's viscosity, we kept a constant number of moles n=1 in the system during our simulations, while the number of moles belonging to the first component was varied from 0 to 1, with the increment 0.1. The system's viscosity was determined after fitting the parabolic velocity profile established across the channel, as done for the single component fluid.

The dynamic viscosity η of the system should equal the sum of the dynamic viscosities η_1 and η_2 of the two components

$$\eta = \eta_1 + \eta_2 \tag{1.40}$$

Since the dynamic viscosity η of a fluid is expressed as the product of its mass density ρ and kinematic viscosity ν

$$\eta = \rho \nu \tag{1.41}$$

we have, for the two component fluid system

$$\rho\nu = (\rho_1 + \rho_2)\nu = \rho_1\nu_1 + \rho_2\nu_2 \tag{1.42}$$

This gives a linear dependence of the kinematic viscosity ν of the whole system on the mass composition ω

$$\nu = \frac{\rho_1}{\rho} \nu_1 + \frac{\rho_2}{\rho} \nu_2 = \omega (\nu_1 - \nu_2) + \nu_2 \tag{1.43}$$

Thus, the slope of the straight line $\nu = \nu(\omega)$ is determined solely by the difference between the kinematic viscosities of each pure component. This slope is positive when $\nu_1 > \nu_2$ and negative otherwise.

Because the kinematic viscosity ν has a qualitatively different dependence on the relaxation time τ_p when using the Upwind or the Lax - Friedrichs schemes (see Figure 1.12), also the composition dependence of the viscosity of a binary system is found to have a different behavior when using these schemes. This is seen in Figure 1.17, where we show the results obtained for a binary system whose particles have the same mass, but the relaxation times τ_p are different for each component. Although the linear interpolation schemes (Upwind and Lax - Friedrichs schemes) do not give the correct dependence of the kinematic viscosity (1.36), the mass concentration dependence of the system viscosity is always linear, as predicted by Eq. (1.43) and shown in Figure 1.18.

Figures 1.19 and 1.20 show some results obtained with the ISLB-CP scheme. Similar graphs were obtained using the ISLB-PC scheme. The correct dependence of the viscosity of the whole system on the mass composition, as well as the other simulation parameters (masses m_1 and m_2 of the particles belonfing to each component, time step δt as well as relaxation times τ_1 and τ_2 of each component), is always recovered.

We should point here that the former LGLB model cannot account for results similar to those shown in figure 1.19 for a homogeneous two component system. In the LGLB model, all particles share the same thermal velocity, even if their masses are different. Since the viscosity in the LGLB model is mass independent, the use of this model to simulate a two component fluid system will always give a composition independent viscosity. This erroneous behavior of the LGLB model is limiting seriously its application to multicomponent fluid systems.

To compare the CS and the ISLB-CP schemes, we used the lattice spacing $\delta x = 0.0001$ m and the time step $\delta t = 10^{-9}$ s to study the composition dependence of the viscosity of a very particular two component system. The two components of this system had the masses $m_1 = 1$ amu, $m_2 = 2$ amu and the relaxation times $\tau_1 = 1 \times 10^{-8}$ s, $\tau_2 = 2 \times 10^{-8}$ s, respectively. In accordance to the viscosity formula (1.37), the viscosities of the two component fluids are identical when we use the CS scheme, since, for this particular system, we have

$$\frac{\tau_1}{m_1} = \frac{\tau_2}{m_2} \tag{1.44}$$

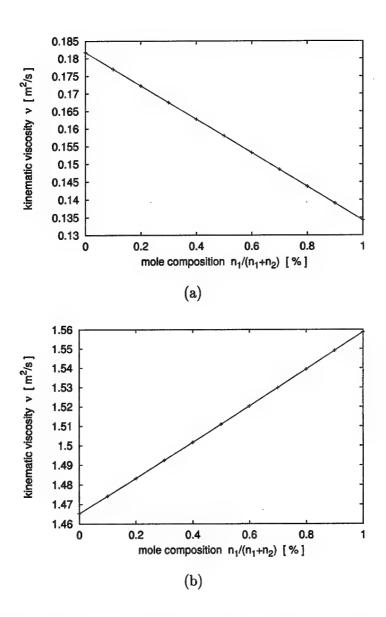


Figure 1.17: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid: (a) - Upwind scheme; (b) - Lax - Friedrichs scheme. The two components have the same mass (1 amu) but the relaxation times are different ($\tau_1 = 10^{-8}$ s and $\tau_2 = 2 \times 10^{-8}$ s).

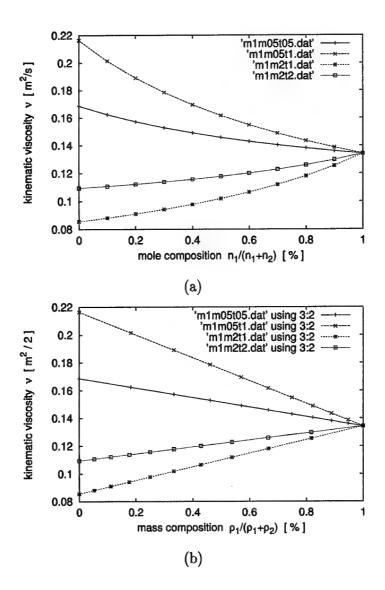


Figure 1.18: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (Upwind scheme). The two components have different masses (expressed in amu) and/or relaxation times (expressed in s). Curves were been obtained for $m_1=1, m_2=0.5, \tau_1=1\times 10^{-8}, \tau_2=0.5\times 10^{-8}$, (upper curve), $m_1=1, m_2=0.5, \tau_1=1\times 10^{-8}, \tau_2=1\times 10^{-8}$, (second curve), $m_1=1, m_2=2, \tau_1=1\times 10^{-8}, \tau_2=1\times 10^{-8}$, (third curve), $m_1=1, m_2=2, \tau_1=1\times 10^{-8}, \tau_2=2\times 10^{-8}$, (bottom curve).

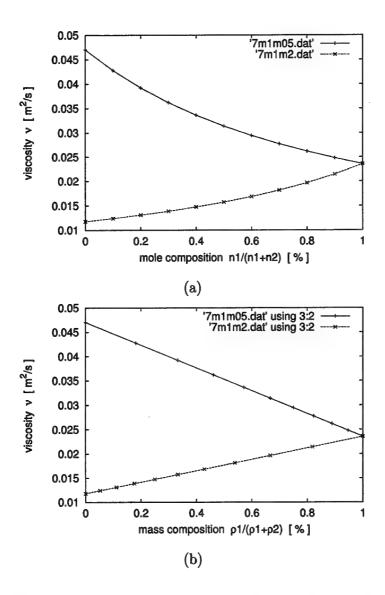


Figure 1.19: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (ISLB-CP scheme). The two components have different masses (expressed in amu) and the same relaxation times (expressed in s). Curves were been obtained for $m_1=1, m_2=0.5, \tau_1=1\times 10^{-8}, \tau_2=1\times 10^{-8},$ (upper curve) and $m_1=1, m_2=2, \tau_1=1\times 10^{-8}, \tau_2=1\times 10^{-8},$ (lower curve), respectively.

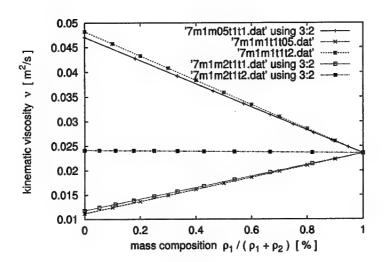


Figure 1.20: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (ISLB-CP scheme). Curves (from top to bottom) were been obtained for (masses are expressed in amu, while relaxation times are expressed in seconds): $m_1 = 1, m_2 = 1, \tau_1 = 1 \times 10^{-8}, \tau_2 = 2 \times 10^{-8}, m_1 = 1, m_2 = 0.5, \tau_1 = 1 \times 10^{-8}, \tau_2 = 1 \times 10^{-8}, m_1 = 1, m_2 = 2, \tau_1 = 1 \times 10^{-8}, \tau_2 = 2 \times 10^{-8}, m_1 = 1, m_2 = 1, \tau_1 = 1 \times 10^{-8}, \tau_2 = 0.5 \times 10^{-8}, m_1 = 1, m_2 = 2, \tau_1 = 1 \times 10^{-8}, \tau_2 = 2 \times 10^{-8}.$

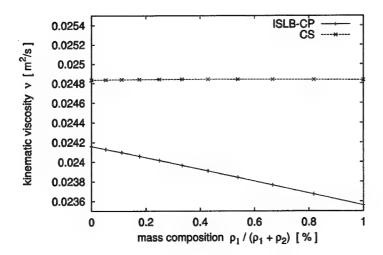


Figure 1.21: Composition dependence of the kinematic viscosity ν of a homogeneous two component fluid (ISLB-CP and CS schemes). Curves were been obtained for $m_1 = 1$ amu, $m_2 = 2$ amu, $\tau_1 = 1 \times 10^{-8}$ s, $\tau_2 = 2 \times 10^{-8}$ s.

When we use the ISLB-CP scheme (as well as the ISLB-PC scheme), the viscosities of the two fluid components are no more equal since

$$\frac{2\tau_1 - \delta t}{m_1} \neq \frac{2\tau_2 - \delta t}{m_2} \tag{1.45}$$

Thus, the viscosity of the particular homogeneous two component system mentioned above is composition independent when the CS scheme is used, while it does not have this property when the ISLB schemes (ISPB-CP or ISLB-PC) are used instead. Figure 1.21 shows the computer results.

Chapter 2

Diffusion Couple

2.1 Description of the model

We consider a two component system whose particles carry different masses denoted m_{σ} , $\sigma=1, 2$. The system is described by two distribution functions sets $\{f_i^{\sigma,N}(\mathbf{r},t)\}$, $\sigma=1, 2; i=0, 1 \dots N$, defined on a 2D regular lattice. The distribution function $f_i^{\sigma,N}(\mathbf{r},t)$ gives the probability to find in node \mathbf{r} of the lattice, at time t, a particle of mass m_{σ} having the velocity $\mathbf{e}_i^{\sigma,N}$, $i=0,1\dots N$. As discussed previously [1], the underlying lattice may be a square one, while the velocity space may be restricted to one of the two 2D discrete velocity sets $\{\mathbf{e}_i^{\sigma,N}\}$ widely used in LB simulations, which correspond to N=6 or N=8. These velocity sets define the so-called seven bit LB model (N=6)

$$\mathbf{e}_{0}^{\sigma,6} = 0$$

$$\mathbf{e}_{i}^{\sigma,6} \equiv \left(e_{i1}^{\sigma,6}, e_{i2}^{\sigma,6}\right) = \left[\cos\frac{2\pi(i-1)}{6}, \sin\frac{2\pi(i-1)}{6}\right] c_{\sigma,6}$$

$$i = 1, \dots 6$$
(2.1)

or the nine bit model (N = 8)

$$e_0^{\sigma,8} = 0$$
 (2.2)
 $e_i^{\sigma,8} = \left[\cos\frac{\pi(i-1)}{2}, \sin\frac{\pi(i-1)}{2}\right] c_{\sigma,8}, \quad i = 1, \dots 4$

$$\mathbf{e}_{i}^{\sigma,8} = \sqrt{2} \left[\cos \left(\frac{\pi}{4} + \frac{\pi(i-5)}{2} \right), \sin \left(\frac{\pi}{4} + \frac{\pi(i-5)}{2} \right) \right] c_{\sigma,8}$$

$$i = 5, \dots 8$$

The "thermal" velocities $c_{\sigma,N}$, $N \in \{6,8\}$ are defined by

$$c_{\sigma,N} = \sqrt{\frac{k_B T}{m_\sigma \chi_N}} \tag{2.3}$$

where

$$\chi_N = \begin{cases} \frac{1}{4} & , & N = 6 \\ \frac{1}{3} & , & N = 8 \end{cases}$$
 (2.4)

In the absence of external forces, the distribution functions are supposed to satisfy the Boltzmann equation with the Bhatnagar - Gross - Krook collision term $(\partial_t = \partial/\partial t, \partial_\alpha = \partial/\partial x_\alpha, \alpha = 1, 2)$

$$\partial_t f_i^{\sigma,N}(\mathbf{r},t) + e_{i\alpha}^{\sigma,N} \partial_{\alpha} f_i^{\sigma,N}(\mathbf{r},t) = -\frac{1}{\tau_{\sigma}} \left[f_i^{\sigma,N}(\mathbf{r},t) - f_i^{\sigma,N,eq}(\mathbf{r},t) \right]$$

$$\sigma = 1, 2 \quad \text{and} \quad i = 0, 1, \dots N$$
(2.5)

and therefore evolve in accordance to the Finite Difference Lattice Boltzmann (FDLB) equations

$$f_{i}^{\sigma,N}(\mathbf{r},t+\delta t) \simeq f_{i}^{\sigma,N}(\mathbf{r},t) - \delta t \, \mathbf{e}_{i}^{\sigma,N} \cdot \nabla_{\mathbf{r}} f_{i}^{\sigma,N}(\mathbf{r},t)$$

$$- \frac{\delta t}{\tau_{\sigma}} \left[f_{i}^{\sigma,N}(\mathbf{r},t) - f_{i}^{\sigma,N,eq}(\mathbf{r},t) \right]$$

$$\sigma = 1, 2 \quad \text{and} \quad i = 0, 1, \dots N$$
(2.6)

where τ_{σ} , $\sigma=1$, 2 are physical relaxation times and $f_i^{\sigma,N,eq}(\mathbf{r},t)$ are the equilibrium distribution functions.

The local number densities are defined by

$$n_{\sigma}(\mathbf{r},t) = \sum_{i=0}^{N} f_{i}^{\sigma,N}(\mathbf{r},t) = \sum_{i=0}^{N} f_{i}^{\sigma,N,eq}(\mathbf{r},t)$$
 (2.7)

while the local density and the local velocity of each component are

$$\rho_{\sigma}(\mathbf{r},t) = m_{\sigma}n_{\sigma}(\mathbf{r},t) \tag{2.8}$$

$$\mathbf{u}_{\sigma}(\mathbf{r},t) = \frac{1}{n_{\sigma}(\mathbf{r},t)} \sum_{i=1}^{N} \mathbf{e}_{i}^{\sigma,N} f_{i}^{\sigma,N}(\mathbf{r},t)$$
 (2.9)

The mass averaged (barycentric) velocity is defined by

$$\mathbf{u} = \frac{m_1 n_1 \mathbf{u}_1 + m_2 n_2 \mathbf{u}_2}{m_1 n_1 + m_2 n_2} = \frac{\sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} \mathbf{u}_{\sigma}}{\sum_{\sigma=1}^{\sigma=2} \rho_{\sigma}}$$
(2.10)

The mass flux \mathbf{j}_{σ} of the component σ [12, 13] is defined with respect to the barycentric velocity

$$\mathbf{j}_{\sigma} = rho_{\sigma} \left(\mathbf{u}_{\sigma} - \mathbf{u} \right) \tag{2.11}$$

As usual in LB models, the equilibrium density functions are expressed as power series in the components of the local equilibrium velocity, $\mathbf{u}^{\sigma,N,eq}$

$$f_{i}^{\sigma,N,eq} = w_{i}^{N} n_{\sigma}(\mathbf{r},t)$$

$$\times \left[1 + \frac{\mathbf{e}_{i}^{\sigma,N} \cdot \mathbf{u}^{\sigma,N,eq}}{\chi_{N} c_{\sigma,N}^{2}} + \frac{(\mathbf{e}_{i}^{\sigma,N} \cdot \mathbf{u}^{\sigma,N,eq})^{2}}{2\chi_{N} c_{\sigma,N}^{4}} - \frac{(\mathbf{u}^{\sigma,N,eq})^{2}}{2\chi_{N} c_{\sigma,N}^{2}} \right]$$

$$(2.12)$$

where the weight factors w_i^N are

$$w_i^6 = \begin{cases} \frac{1}{2} & , & i = 0\\ \frac{1}{12} & , & i = 1, \dots, 6 \end{cases}$$
 (2.13)

for the seven bit model, and

$$w_i^8 = \begin{cases} \frac{4}{9} & , & i = 0\\ \frac{1}{9} & , & i = 1, \dots 4\\ \frac{1}{36} & , & i = 5, \dots, 8 \end{cases}$$
 (2.14)

for the nine bit model.

Since the elements of the velocity sets $\{\mathbf{e}_i^{\sigma,N}\}$ satisfy

$$\sum_{i} e_{i\alpha}^{\sigma,N} = 0$$

$$\sum_{i} w_{i}^{N} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} = \chi_{\sigma} c_{\sigma,N}^{2} \delta_{\alpha\beta}$$

$$\sum_{i} w_{i}^{N} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} = 0$$

$$\sum_{i} w_{i}^{\sigma,N} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} = \chi_{\sigma}^{2} c_{\sigma,N}^{4} (\delta_{\alpha\beta} \delta_{\gamma\delta} + \delta_{\gamma\beta} \delta_{\alpha\delta} + \delta_{\delta\beta} \delta_{\gamma\alpha})$$

$$\sum_{i} w_{i}^{\sigma,N} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} e_{i\delta}^{\sigma,N} = \chi_{\sigma}^{2} c_{\sigma,N}^{4} (\delta_{\alpha\beta} \delta_{\gamma\delta} + \delta_{\gamma\beta} \delta_{\alpha\delta} + \delta_{\delta\beta} \delta_{\gamma\alpha})$$

following sums are easily computed

$$\sum_{i=0}^{i=\sigma} f_i^{\sigma,N,eq} = n_{\sigma} \tag{2.16}$$

$$\sum_{i=0}^{i=\sigma} e_{i\alpha}^{\sigma,N} f_i^{\sigma,N,eq} = n_{\sigma}(\mathbf{r},t) u_{\alpha}^{\sigma,N,eq}$$
(2.17)

$$\sum_{i=0}^{i=\sigma} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma}, f_i^{\sigma,N,eq} = n_{\sigma} \left[\chi_{\sigma} c_{\sigma}^2 \delta_{\alpha\beta} + u_{\alpha} u_{\beta} \right]$$
 (2.18)

$$\sum_{i=0}^{i=\sigma} e_{i\alpha}^{\sigma,N} e_{i\beta}^{\sigma,N} e_{i\gamma}^{\sigma,N} f_i^{\sigma,N,eq} = \chi_{\sigma} c_{\sigma,N}^2 n_{\sigma} (\delta_{\alpha\beta} u_{\gamma} + \delta_{\alpha\gamma} u_{\beta} + \delta_{\beta\gamma} u_{\alpha})$$
 (2.19)

As discussed previously [1], the macroscopic behavior of a single component fluid system is independent of the discretization of the phase space (i.e., independent of the number of elements N of the discrete velocity set $\{e_i^{\sigma,N}\}$ used in the FDLB model). In the case of a two component system, we will consider the same model (seven bit of nine bit, i.e., the same N) for both components. The choice N=6 or N=8 is only a technical detail related to the FDLB computer code and has no influence on the results at the macroscopic scale. To reduce the number of indices in the following formulae in this chapter (as well as in the subsequent chapters of this report), we will discard the index N. Consequently, we will write

$$\partial_t f_i^{\sigma}(\mathbf{r},t) + e_{i\alpha}^{\sigma} \partial_{\alpha} f_i^{\sigma}(\mathbf{r},t) = -\frac{1}{\tau_{\sigma}} \left[f_i^{\sigma}(\mathbf{r},t) - f_i^{\sigma,eq}(\mathbf{r},t) \right]$$

$$\sigma = 1, 2$$
 and $i = 0, 1, \dots N$ (2.20)

instead of Eq. (2.5),

$$f_i^{\sigma,eq} = w_i n_{\sigma}(\mathbf{r},t) \left[1 + \frac{\mathbf{e}_i^{\sigma} \cdot \mathbf{u}^{\sigma,eq}}{\chi_{\sigma} c_{\sigma}^2} + \frac{(\mathbf{e}_i^{\sigma} \cdot \mathbf{u}^{\sigma,eq})^2}{2\chi_{\sigma}^2 c_{\sigma}^4} - \frac{(\mathbf{u}^{\sigma,eq})^2}{2\chi_{\sigma} c_{\sigma}^2} \right]$$
(2.21)

instead of Eq. (2.13), and so on.

2.2 Chapman - Enskog expansion

The distribution functions are usually expanded as power series in the Knudsen number ε

 $f_i^{\sigma} = f_i^{\sigma(0)} + \varepsilon f_i^{\sigma(1)} + \varepsilon^2 f_i^{\sigma(2)} + \dots$ (2.22)

while two corresponding time scales and one length scale are introduced when computing time and space derivatives

$$\partial_t \equiv \varepsilon \partial_{t_1} + \varepsilon^2 \partial_{t_2} \tag{2.23}$$

$$\partial_{\alpha} \equiv \varepsilon \partial_{\alpha} \tag{2.24}$$

After introducing the Chapman - Enskog expansion (2.22), as well as the corresponding expansions (2.23) and (2.24) of the time and space derivatives in the Boltzmann equation (2.20)

$$\left(\varepsilon\partial_{t_{1}} + \varepsilon^{2}\partial_{t_{2}}\right)\left[f_{i}^{\sigma(0)} + \varepsilon f_{i}^{\sigma(1)} + \varepsilon^{2} f_{i}^{\sigma(2)} + \ldots\right] + \\ \varepsilon e_{i\beta}^{\sigma}\partial_{\beta}\left[f_{i}^{\sigma(0)} + \varepsilon f_{i}^{\sigma(1)} + \varepsilon^{2} f_{i}^{\sigma(2)} + \ldots\right] = \\ -\frac{1}{\tau_{\sigma}}\left[f_{i}^{\sigma(0)} + \varepsilon f_{i}^{\sigma(1)} + \varepsilon^{2} f_{i}^{\sigma(2)} + \ldots - f_{i}^{\sigma,eq}\right]$$
(2.25)

we can separate the zero-th, first and second order Boltzmann equations with respect to the Knudsen number ε

$$0 = -\frac{1}{\tau_{\sigma}} \left[f_i^{\sigma(0)} - f_i^{\sigma,eq} \right] \quad (2.26)$$

$$\partial_{t_1} f_i^{\sigma(0)} + \partial_{\beta} f_i^{\sigma(0)} e_{i\beta}^{\sigma} = -\frac{1}{\tau_{\sigma}} f_i^{\sigma(1)}$$
 (2.27)

$$\partial_{t_2} f_i^{\sigma(0)} + \partial_{t_1} f_i^{\sigma(1)} + \partial_{\beta} f_i^{\sigma(1)} e_{i\beta}^{\sigma} = -\frac{1}{\tau_{\sigma}} f_i^{\sigma(2)}$$
 (2.28)

From the zero-th order Boltzmann equation (2.26), we get, for any $\sigma \in \{1,2\}$ and $i=0,1\ldots N$

 $f_i^{\sigma(0)} = f_i^{\sigma,eq} \tag{2.29}$

This condition ensures also the validity of the zero-th order mass and momentum equations (2.32) and (2.35), which will be discussed in the next section.

Taking into account the expression (2.7) of the local number density n_{σ} and the series expansion (2.22), we get

$$\sum_{i=0}^{i=N} f_i^{\sigma(l)} = 0 \qquad \forall l \ge 1 \tag{2.30}$$

As usual in many Lattice Boltzmann models for multicomponent fluids [13, 14, 15, 16, 17], we assume that all components have the same equilibrium velocity in the absence of external forces and long range interactions

$$u_{\alpha}^{\sigma,eq} \equiv u_{\alpha}' \quad \forall \sigma = 1,2 \quad \text{and} \quad \alpha = 1,2$$
 (2.31)

Consequently, the equilibrium distribution functions $f_i^{\sigma,eq} \equiv f_i^{\sigma(0)}$ are expressed as series expansions (2.21) with respect to the components u'_{α} of this equilibrium velocity. At this stage, we are not interested in the procedure which should provide the means to compute the values of the components u'_{α} ; this procedure will be discussed later.

2.3 Conservation equations

The zero-th, first and second order mass conservation equations for each component $\sigma = 1$, 2 are recovered from Eqs. (2.26 – 2.28) after summation with respect to i (multiplication by m_{σ} is omitted)

$$-\frac{1}{\tau_{\sigma}} \sum_{i=0}^{i=N} \left[f_i^{\sigma(0)} - f_i^{\sigma,eq} \right] = 0$$
 (2.32)

$$\partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_{\beta} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\beta}^{\sigma} = -\frac{1}{\tau_{\sigma}} \sum_{i=0}^{i=N} f_i^{\sigma(1)} (2.33)$$

$$\partial_{t_2} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(1)} + \partial_{\beta} \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\beta}^{\sigma} = -\frac{1}{\tau_{\sigma}} \sum_{i=0}^{i=N} f_i^{\sigma(2)} (2.34)$$

The zero-th, first and second order momentum conservation equations are also recovered from Eqs. (2.26 – 2.28) after multiplication with $m_{\sigma}e_{i\alpha}^{\sigma}$ and summation with respect to i and σ

$$-\sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma}}{\tau_{\sigma}} \sum_{i=0}^{i=N} \left[f_{i}^{\sigma(0)} - f_{i}^{\sigma,eq} \right] e_{i\alpha}^{\sigma} = 0 \quad (2.35)$$

$$\partial_{t_{1}} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_{i}^{\sigma(0)} e_{i\alpha}^{\sigma} + \partial_{\beta} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_{i}^{\sigma(0)} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} = -\sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma}}{\tau_{\sigma}} \sum_{i=0}^{i=N} f_{i}^{\sigma(1)} e_{i\alpha}^{\sigma}$$
(2.36)

$$\partial_{t_2} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^{\sigma} + \partial_{t_1} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\alpha}^{\sigma} +$$

$$\partial_{\beta} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_{i}^{\sigma(1)} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} = -\sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma}}{\tau_{\sigma}} \sum_{i=0}^{i=N} f_{i}^{\sigma(2)} e_{i\alpha}^{\sigma}$$
(2.37)

The zero-th order mass and momentum equations (2.32) and (2.35) are automatically satisfied in accordance to Eq. (2.29). Moreover, the right hand sides of the first and second order mass equations (2.33) and (2.34) vanish because of Eq. (2.30) and thus, these equations are rewritten as

$$\partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_{\beta} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\beta}^{\sigma} = 0$$
 (2.38)

$$\partial_{t_2} \sum_{i=0}^{i=N} f_i^{\sigma(0)} + \partial_{t_1} \sum_{i=0}^{i=N} f_i^{\sigma(1)} + \partial_{\beta} \sum_{i=0}^{i=N} f_i^{\sigma(1)} e_{i\beta}^{\sigma} = 0$$
 (2.39)

2.4 Mass equations

Since $f_i^{\sigma(0)} = f_i^{\sigma,eq}$, we can use the series expansion (2.21), as well as the properties (2.16) and (2.17), to rewrite the first order mass conservation equation (2.38) in a more familiar form, after multiplication by m_{σ}

$$\partial_{t_{1}}\rho_{\sigma} + \partial_{\beta}(\rho_{\sigma}u_{\beta}') = 0 \tag{2.40}$$

Here

$$\rho_{\sigma} = m_{\sigma} n_{\sigma} \tag{2.41}$$

is the local density of the component σ .

Using the first order Boltzmann equation (2.27) to express $f_i^{\sigma(1)}$ in the second order equation (2.28), we get

$$\partial_{t_2} f_i^{\sigma(0)} + au_{\sigma} (\partial_{t_1})^2 f_i^{\sigma(0)} - 2 au_{\sigma} \, \partial_{t_1} \left[\partial_{t_1} f^{\sigma(0)} + \partial_{eta} f_i^{\sigma(0)} e_{ieta}^{\sigma} \right] -$$

$$\tau_{\sigma} \partial_{\alpha} \partial_{\beta} f_{i}^{\sigma(0)} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} = \frac{1}{\tau_{\sigma}} f_{i}^{\sigma(2)}$$
 (2.42)

This equation, multiplied by m_{σ} and summed over i, gives the following form of the the second order mass conservation equation (2.39)

$$\partial_{t_2}\rho_{\sigma} + \tau_{\sigma}(\partial_{t_1})^2 \rho_{\sigma} - \tau_{\sigma}\partial_{\alpha}\partial_{\beta}(n_{\sigma}k_B T \delta_{\alpha\beta} + \rho_{\sigma}u_{\alpha}'u_{\beta}') = 0$$
 (2.43)

To derive this result, we used the first order mass equation (2.40), as well as the property (2.18) and the definition (2.3).

To recover the mass conservation equation for the component σ up to the second order with respect to the Knudsen number, we sum together Eqs. (2.40) and (2.43) multiplied by ε and ε^2 , respectively, and take into account the expressions (2.23) and (2.24) of the time and space derivatives

$$\partial_{t}\rho_{\sigma} + \partial_{\alpha}(\rho_{\sigma}u_{\alpha}') + \tau_{\sigma}(\partial_{t})^{2}\rho_{\sigma} -$$

$$\tau_{\sigma} \frac{k_{B}T}{m_{\sigma}} \delta_{\alpha\beta}\partial_{\alpha}\partial_{\beta}\rho_{\sigma} - \tau_{\sigma}\partial_{\alpha}\partial_{\beta}(\rho_{\sigma}u_{\alpha}'u_{\beta}') = 0$$
(2.44)

When summing over σ , we get also the mass equation for the whole fluid system

$$\partial_t \rho + \partial_{\alpha} (\rho u_{\alpha}') + (\partial_t)^2 \sum_{\sigma=1}^{\sigma=2} \tau_{\sigma} \rho_{\sigma} -$$

$$k_B T \, \partial_{\alpha} \partial_{\beta} \sum_{\sigma=1}^{\sigma=2} \frac{\tau_{\sigma} \rho_{\sigma}}{m_{\sigma}} \, - \, \partial_{\alpha} \partial_{\beta} \left[\left(\sum_{i=1}^{i=2} \tau_{\sigma} \rho_{\sigma} \right) u_{\alpha} u_{\beta} \right] = 0 \qquad (2.45)$$

where

$$\rho = \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} \tag{2.46}$$

is the local density of the whole fluid. In the particular case $\tau_{\sigma} = \tau \, \forall \sigma$, the mass equation for the whole fluid (2.45) becomes

$$\partial_t \rho + \partial_\alpha (\rho u_\alpha') + \tau (\partial_t)^2 \rho_\sigma - \tau \partial_\alpha \partial_\beta p - \tau \partial_\alpha \partial_\beta \left(\rho u_\alpha' u_\beta' \right) = 0 \qquad (2.47)$$

where

$$p = k_B T \sum_{\sigma=1}^{\sigma=N} n_{\sigma} \tag{2.48}$$

is the local pressure of the fluid system, which is a mixture of two ideal gases.

The mass equation for a single component (2.44), as well as the mass equation for the whole system (2.45) contain the first and second order time derivatives of the local density. The presence of both time derivatives is a characteristics of the telegraphist equation [18], which describes a propagation phenomenon. Consequently, one may expect that the density profile across the system exhibits a kink (wrinkle or peak), i.e., a non monotonic propagating pulse. This behavior should be dominant for small values of the product between the local density and the equilibrium velocity ($\rho_{\sigma}u'_{\alpha} \simeq 0$), when the mass equation for the component σ reduces to the true telegraphist equation

$$\partial_t \rho_\sigma + \frac{\mathcal{D}}{C} (\partial_t)^2 \rho_\sigma = \mathcal{D} \nabla^2 \rho_\sigma \tag{2.49}$$

where the diffusion coefficient is

$$\mathcal{D} = \tau_{\sigma} \frac{k_B T}{m_{\sigma}} \tag{2.50}$$

and

$$C = \frac{k_B T}{m_\sigma} \tag{2.51}$$

is the finite speed at which information travels in the system. The diffusion equation is recovered in the case

$$\frac{\mathcal{D}}{\mathcal{C}} = \tau_{\sigma} \to 0 \tag{2.52}$$

2.5 Equilibrium velocity u'

The right hand side of the Boltzmann equation (2.20) represents the collision term which expresses the variation of the equilibrium distribution function as a result of collisions between particles. Collisions should preserve the local momentum of the whole system. Since the momentum equation is derived from the Boltzmann equation after multiplication by $m_{\sigma}e_{i\alpha}^{\sigma}$ and summation over the indices σ and i, we should have

$$\sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma}}{\tau_{\sigma}} \sum_{i=1}^{i=N} \left[f_{i}^{\sigma} - f_{i}^{\sigma,eq} \right] e_{i\alpha}^{\sigma} = 0$$
 (2.53)

Since $f_i^{\sigma,eq}$ is expressed as a series expansion (2.21 in the equilibrium velocity u'_{α} (which is supposed to be the same for all σ), we can use Eq. (2.17) to get

$$u_{\alpha}' \sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma} n_{\sigma}}{\tau_{\sigma}} = \sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma}}{\tau_{\sigma}} \sum_{i=1}^{i=N} f_{i}^{\sigma} e_{i\alpha}^{\sigma}$$
 (2.54)

which allows the determination of the components u'_{α} of the equilibrium velocity in the node **r** of the lattice.

If we take into account the fact that the distribution functions f_i^{σ} are expressed as power series (2.22) in the Knudsen number, as well as the fact that $f_i^{\sigma(0)} = f_i^{\sigma,eq}$, we get the following relations from Eq. (2.54)

$$\sum_{\sigma=1}^{\sigma=2} \frac{m_{\sigma}}{\tau_{\sigma}} \sum_{i=1}^{i=N} f_i^{\sigma(l)} e_{i\alpha}^{\sigma} = 0 \qquad , \forall l \ge 1$$
 (2.55)

2.6 Momentum equation

If we use Eq. (2.55), the first order momentum equation (2.36) becomes

$$\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^{\sigma} + \partial_{\beta} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} = 0$$
 (2.56)

Introducing the relations (2.17) and (2.18) in the above equation, we get the Euler equation

$$\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' + \partial_{\alpha} p + \partial_{\beta} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' u_{\beta}' = 0$$
 (2.57)

Using the first order Boltzmann equation (2.27) to express $f_i^{\sigma(1)}$ in the second order momentum equation (2.37), we get

$$\partial_{t_2} \sum_{\sigma=1}^{\sigma=2} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^{\sigma} + (\partial_{t_1})^2 \sum_{\sigma=1}^{\sigma=2} \tau_{\sigma} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^{\sigma} -$$

$$2\partial_{t_1} \left[\partial_{t_1} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' + \partial_{\alpha} p + \partial_{\beta} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' u_{\beta}' \right] -$$

$$\partial_{\beta} \partial_{\gamma} \sum_{i=1}^{i=2} \tau_{\sigma} m_{\sigma} \sum_{i=0}^{i=N} f_i^{\sigma(0)} e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} e_{i\gamma}^{\sigma} = 0 \quad (2.58)$$

The square bracket in the equation above vanishes in accordance to Eq. (2.57). If we take into account Eqs. (2.3) and (2.19) and consider $\tau_1 = \tau_2 = \tau$, the second order momentum equation becomes

$$\partial_{t_2} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' + \tau (\partial_{t_1})^2 \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}'$$

$$- \tau k_B T \sum_{\sigma=1}^{\sigma=2} (2\partial_{\alpha} \nabla \cdot \mathbf{u}' + \nabla^2 u_{\alpha}') = 0 \qquad (2.59)$$

When adding the first and second ordet momentum equations, we get

$$\partial_{t} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' + \tau (\partial_{t})^{2} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' + \partial_{\alpha} p + \partial_{\beta} \sum_{\sigma=1}^{\sigma=2} \rho_{\sigma} u_{\alpha}' u_{\beta}'$$
$$- \tau k_{B} T \sum_{\sigma=1}^{\sigma=2} \left[2 \partial_{\alpha} (\nabla \cdot \mathbf{u}') + \nabla^{2} u_{\alpha}' \right] = 0 \qquad (2.60)$$

Thus, the momentum equation up to the second order contains the first and second order time derivatives, like the mass equation of the whole fluid system (2.45. As mentioned previously, the presence of the second order time derivative is a characteristics of a propagation phenomenon [18].

2.7 Pure diffusion

We suppose that the fluid system is always at rest. In this case, the equilibrium velocity u'_{α} should vanish everywhere. When

$$u_{\alpha}' = 0 \tag{2.61}$$

the series expansion (2.21) reduces to

$$f_i^{\sigma(0)} = f_i^{\sigma,eq} = w_i n_\sigma \tag{2.62}$$

Consequently, the first order mass equations (2.38) becomes

$$\partial_{t_1} \sum_{i=0}^{i=N} w_i n_{\sigma} + \partial_{\beta} \sum_{i=0}^{i=N} w_i n_{\sigma} e_{i\beta}^{\sigma} = 0$$
 (2.63)

In accordance to Eqs. (2.13) and (2.14), we have the following equalities

$$\sum_{i=0}^{i=N} w_i = 1 (2.64)$$

$$\sum_{i=0}^{i=N} w_i e_{i\beta}^{\sigma} = 0 (2.65)$$

which are valid for N=6, as well as for N=8 (i.e., independent of the number of the discrete velocity speeds in the phase space). Thus, the first order mass equation (2.63) reduces to

$$\partial_{t_1} n_{\sigma} = 0 \tag{2.66}$$

Therefore, from the first order Boltzmann equation (2.27) we get, using the expression (2.62)

$$f_i^{\sigma(1)} = -\tau_\sigma w_i e_{i\alpha}^\sigma \partial_\alpha n_\sigma \tag{2.67}$$

which may be introduced in the second order mass equation (2.39) to give

$$\partial_{t_2} \sum_{i=0}^{i=N} w_i n_{\sigma} - \partial_{t_1} \sum_{i=0}^{i=N} \tau_{\sigma} w_i e_{i\alpha}^{\sigma} \partial_{\alpha} n_{\sigma} - \partial_{\beta} \sum_{i=0}^{i=N} \tau_{\sigma} w_i e_{i\alpha}^{\sigma} \partial_{\alpha} n_{\sigma} = 0 \quad (2.68)$$

Since

$$\sum_{i=0}^{i=N} w_i e_{i\alpha}^{\sigma} e_{i\beta}^{\sigma} = \chi_{\sigma} c_{\sigma}^2 \delta \alpha \beta \tag{2.69}$$

the second order equation (2.68) is rewritten as

$$\partial_{t_2} n_{\sigma} = \tau_{\sigma} \chi_{\sigma} c_{\sigma}^2 \nabla n_{\sigma} \tag{2.70}$$

After multiplication of the mass equations (2.63) and (2.70) by ε and ε^2 , respectively, we can sum them together to recover the mass equation up to second order with respect to the Knudsen number

$$\partial_t n_\sigma = \tau_\sigma \chi_\sigma c_\sigma^2 \nabla n_\sigma \tag{2.71}$$

which, after multiplication with m_{σ} , becomes identical to the pure diffusion equation

$$\partial_t \rho_\sigma = \mathcal{D} \, \nabla^2 \rho_\sigma \tag{2.72}$$

where the diffusion coefficient is

$$\mathcal{D} = \tau_{\sigma} \chi_{\sigma} c_{\sigma}^2 = \tau_{\sigma} \frac{k_B T}{m_{\sigma}} \tag{2.73}$$

If we remind the expression of the viscosity in the FDLB model [1]

$$\nu = \tau_{\sigma} \chi_{\sigma} c_{\sigma}^2 = \tau_{\sigma} \frac{k_B T}{m_{\sigma}} \tag{2.74}$$

we see that the Schmidt number Sc [12] is a constant equal to unity in the present FDLB model

 $Sc = \frac{\nu}{\mathcal{D}} = 1 \tag{2.75}$

2.8 Comparison with the theory of Shan and Doolen

The LGLB model previously developed by Shan and Doolen [13, 17] was used to simulate the time evolution of the barycentric velocity profile in a diffusion couple [19]. Here is a brief outline of the main characteristics of this model:

- 1. All components have the same equilibrium velocity \mathbf{u}' , which means that the hypothesis expressed by Eq. (2.31) is used.
- 2. The equilibrium velocity \mathbf{u}' in the absence of external forces is given by Eq (2.54).
- 3. The equilibrium distribution functions $f_i^{\sigma,eq}$ are expressed as series expansions in the equilibrium velocity \mathbf{u}' while the leading order distribution functions $f_i^{\sigma(0)}$ are expressed as series expansions in the local fluid velocity \mathbf{u} , (i.e., the barycentric velocity see [17], as well as [19]); this means that Eq. (2.29) is not valid in the model of Shan and Doolen; in fact, they make no explicit use of the zero-th, first and second order Boltzmann equations (2.26 2.28) as separate entities; we may imagine that the starting point of the model of Shan and Doolen is the sum of the zero-th and first order Boltzmann equation

$$\partial_{t_1} f_i^{\sigma(0)} + \partial_{\beta} f_i^{\sigma(0)} e_{i\beta}^{\sigma} = -\frac{1}{\tau_{\sigma}} \left[f_i^{\sigma(0)} + f_i^{\sigma(1)} - f_i^{\sigma,eq} \right]$$
 (2.76)

which allows $f_i^{\sigma(0)} \neq f_i^{\sigma,eq}$, and so on [19].

- 4. The series expansion for $f_i^{\sigma,eq}$ and $f_i^{\sigma(0)}$ which is valid for N=6, i.e., the six bit model (see [19]) contains the parameter d_{σ} which may be adjusted to allow different values of the diffusivity; this "degree of freedom" does not exist in the FDLB model since the very recent approach of He et al. [20, 21, 22] clearly establishes the value $d_{\sigma}=0.5$ using Gaussian quadrature formulae.
- 5. We mention here that Shan and Doolen give two simulation results (figures 1 and 2 in their paper [13]); these figures refer to equilibrium density profiles in a binary mixture which seem to be correct; since these results refer to systems at equilibrium, we think that the barycentric velocity **u**, as well as the equilibrium velocity **u'** vanish and the pure diffusion case is recovered; this may be a serious argument to consider the model as being valid only for stationary (equilibrium) cases, when there is some competition between diffusion and other phenomena generated by external forces, while the model itself does not account for the true dynamic evolution towards the equilibrium (stationary) case.
- 6. We should mention also the fact that the LGLB model of Shan and Doolen inherits the main disadvantage of the LGLB models, which is the fact that the particle thermal speeds in this model are strictly related to the lattice spacing (in fact, the magnitude of these speeds is strictly the lattice spacing divided by the time step) and thus, any LGLB model does not allow different thermal speed for particles carrying different masses; FDLB models allow different thermal speeds when the particles have differents masses and thus, one may expect these models to be more close to the physical reality.

2.9 Simulation results

We used a 250 \times 5 square lattice with walls placed left and right. The lattice spacing was $\delta x = \delta y = 10^{-4}$ m. Periodic conditions were used at the upper and bottom boundaries. The left half of the lattice was initialized with particles belonging to species $\sigma = 1$ while the right half was initialized with particles belonging to species $\sigma = 2$. When developing the computer code, we used the nine bit model (N = 8) since this model allowed a larger variety of finite differences schemes to be tested (see Chapter 1).

Our first diffusion code used the Second Order Runge Kutta for time integration [1], combined with an Upwind scheme for calculating space derivatives. Later, we developed a code based on the First Order Upwind Scheme (FU) which we described in the previous chapter. Both the Runge Kutta code (Appendix B), as well as the FU code (Appendix A) gave similar results which are described below.

The initial particle number density was set to unity for each component. As a result, the initial density profiles of each component, as well as the total density profile were always similar to those depicted in Figure 2.1, where we used $m_1 = 1$ amu and $m_2 = 0.9$ amu. 1 amu (atomic mass unit) equals 1.661×10^{-27} kg.

The present simulations were done with the same value of the relaxation time: $\tau_{\sigma} = \tau = 10^{-8}$ s, while the time step was chosen to be $\delta t = 10^{-9}$ s. The thermal velocities of each species of particles were calculated in accordance to Eq. (2.3) with T=300 K.

Figure 2.2 shows the typical time evolution of the total density profile. One can see the presence of two kinks: a left propagating kink and a right propagating one. Their evolution is presented in Figures 2.3 and 2.4. The presence of these propagating kinks is not a surprise if we remind the mass equation (2.45) of the whole fluid system, which contains the second order time derivative. As mentioned before, the presence of the second order time derivative is a characteristics of a propagation phenomena.

Figure 2.5 shows the time evolution of the barycentric velocity. We may see the two kinks which form and propagate laterally, while the profile exhibits a central peak which remains always positive. The propagating kinks are reflected by the lateral walls and superpose to the central peak at some moments during their propagation. The formation of these propagating kinks in the figures showing the time evolution of the barycentric velocity may be explained by the presence of the second order time derivative in the momentum equation (2.60).

Figure 2.6 shows the time evolution of the mass flux of component 2. This flux is orientated from left to right, as expected, and does not change its sign. No kinks are observed.

Figure 2.7 shows the total density profile when the two species of particles have the same mass. This profile is found to be constant during the diffusion process, while the individual profiles of components 1 and 2 evolve separately. Figure 2.8 shows the corresponding time evolution of the local density of component 1. One can see that kinks are not present in the den-

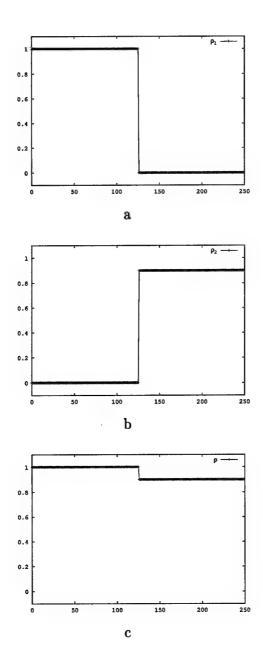


Figure 2.1: Initial density profiles in the diffusion couple: a – component $\sigma=1$; b – component $\sigma=2$; c – total density profile $\rho=\rho_1+\rho_2$.

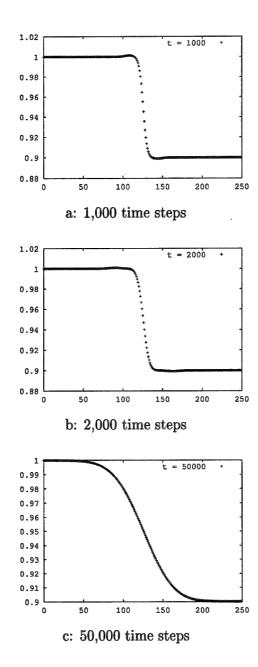


Figure 2.2: Time evolution of the total density profiles in the diffusion couple.

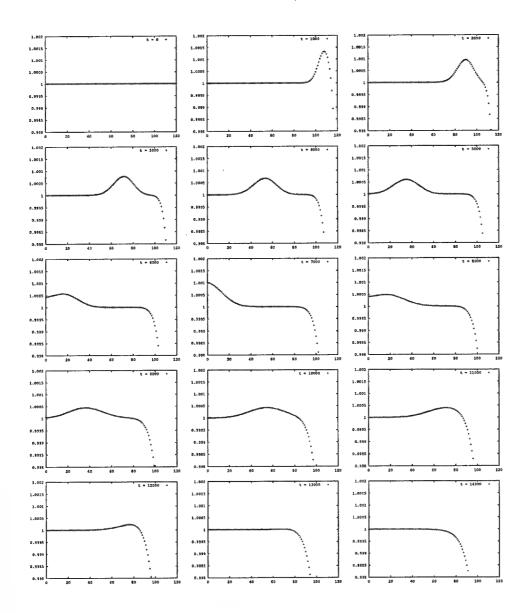


Figure 2.3: Time evolution of the total density profiles in the left region of the diffusion couple (snapshots are taken every 1000 time steps).

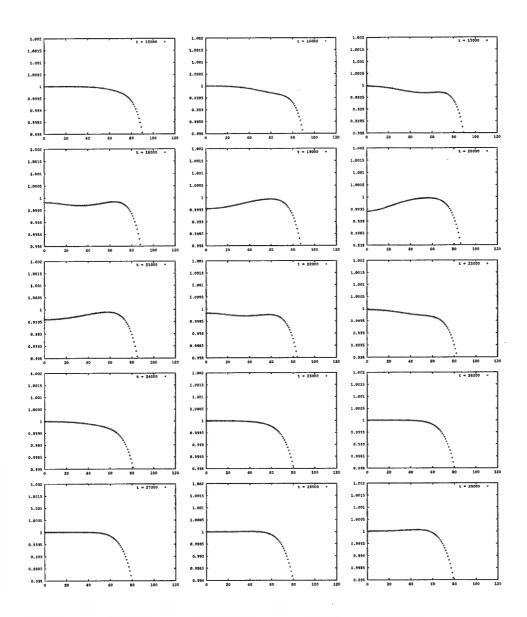


Figure 2.3: (continued) Time evolution of the total density profiles in the left region of the diffusion couple (snapshots are taken every 1000 time steps).

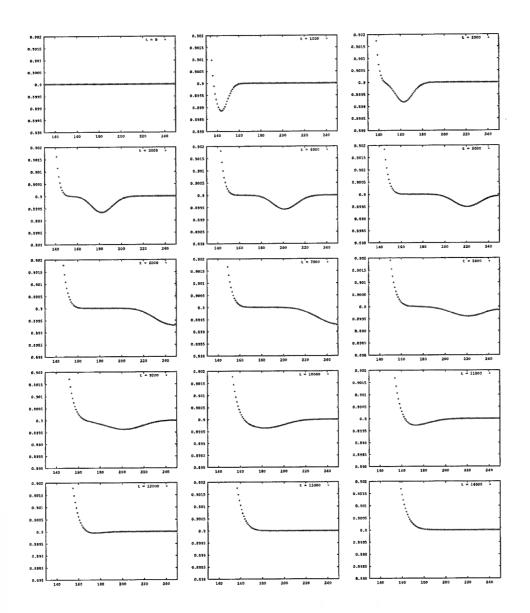


Figure 2.4: Time evolution of the total density profiles in the right region of the diffusion couple (snapshots are taken every 1000 time steps).

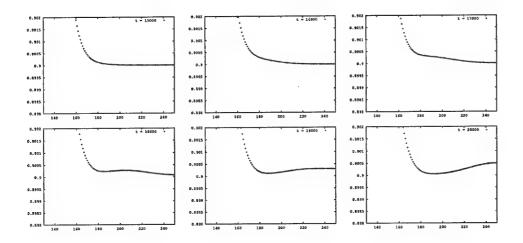


Figure 2.4: (continued) Time evolution of the total density profiles in the right region of the diffusion couple (snapshots are taken every 1000 time steps).

sity profiles when masses are equals, which means that we are dealing with a pure diffusion process.

To compare the FDLB simulations wit the former LGLB simulations [19], we slightly modified the corresponding computer code and set the same thermal velocity for each species of particles, even if the masses are different. For convenience, we used $m_1 = 1$ amu, $m_2 = 0.9$ amu, while c_1 was given by Eq. (2.3) and c_2 was forced to be equal to c_1 . Figure 2.9 shows the time evolution of the barycentric velocity in the diffusion couple, which is very similar to the LGLB results reported in [19]. In this case, the barycentric velocity changes its sign alternatively, due to strong oscillations which are present in the system.

The unphysical behavior of the barycentric velocity reported in figure 2.9, as well as the simulation results reported in [19] provide a very strong reason for a careful handling of the former LGLB models when dealing with multicomponent (e.g., binary) systems whose particles carry different masses. All the present Lattice Boltzmann litterature dealing with multicomponent systems, e.g., the approaches in [11, 14, 18, 23, 24, 25, 26, 27, 28, 29, 30], ignores completely the connection between the lattice spacing and the thermal speeds of particles and do not report unphysical effects of the Lattice Gas like

Lattice Boltzmann scheme. The unphysical oscillations which are observed when trying to simulate the behavior of a diffusion couple in microgravity environment with the LGLB model revealed a severe limitation of these models to those multicomponent systems whose particles have identical mass (as well as the same thermal velocity). Finite Difference Lattice Boltzmann Models (FDLB) or Interpolation Supplemented Lattice Boltzmann Models (ISLB) should be used in order to allow different thermal velocities of particles in the system (i.e., different values of the Courant - Friedrichs - Lewy number CFL). In this respect, the absence of the oscillations of the barycentric velocity in the diffusion couple, as reported in Figure 2.5, are very encouraging.

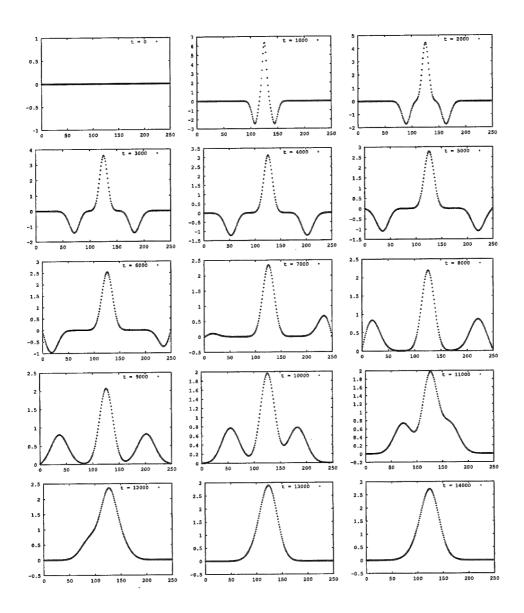


Figure 2.5: Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

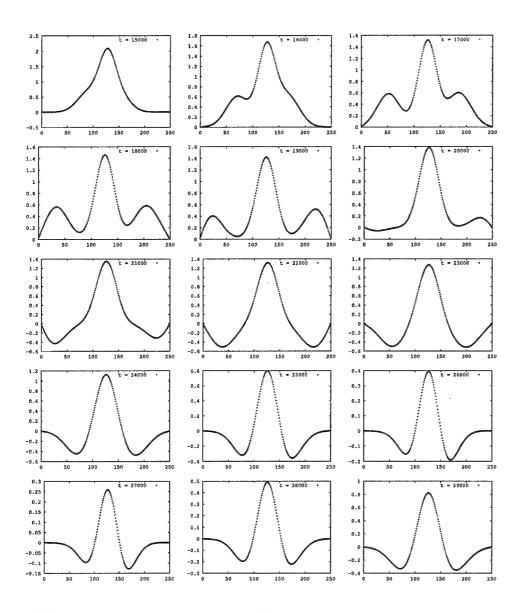


Figure 2.5: (continued) Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

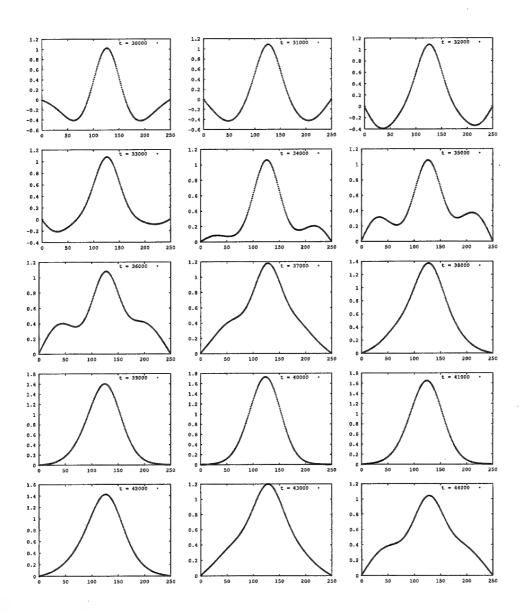


Figure 2.5: (continued) Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

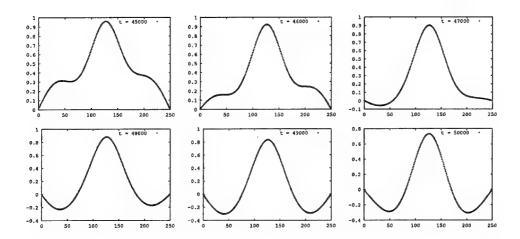


Figure 2.5: (continued) Time evolution of the barycentric velocity profile (snapshots are taken every 1000 time steps).

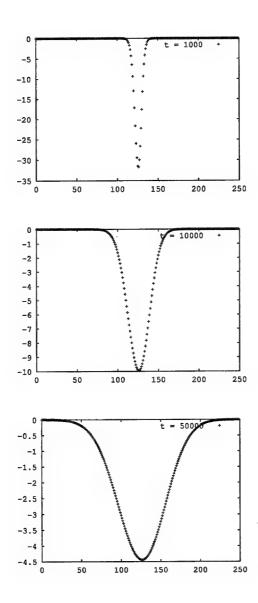


Figure 2.6: Time evolution of mass flux of component 2.

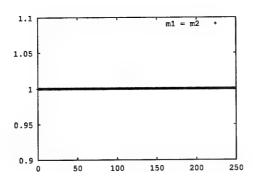


Figure 2.7: Total density profiles in the diffusion couple when the two species of particles have equal mass: this profile remains constant during the diffusion process.

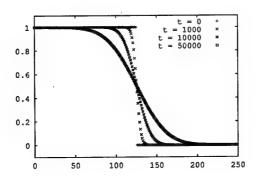


Figure 2.8: Time evolution of the density profiles of component 1 when the two species of particles have equal mass: no kinks are observed.

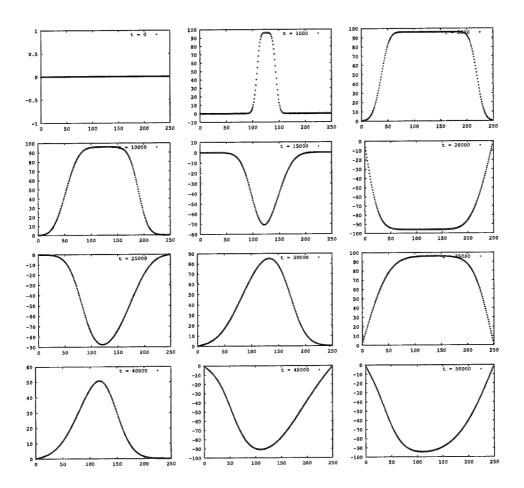


Figure 2.9: Time evolution of the barycentric velocity profile (snapshots are taken every 5000 time steps, except the second one, which is taken after 1000 time steps) when the two thermal velocities are set equal $(c_1 = c_2)$.

Chapter 3

Sessile drop

3.1 Drop shape

A computer code based on a lattice Boltzmann model for liquid - vapor systems was already described in the preliminary report [31]. This code was used to investigate the equilibrium shape of a liquid drop on a horizontal surface and some preliminary results were presented in [31]. These preliminary simulations were done on a 256×64 lattice because of limitations in the computer resources available at that time. In the present report we will refer to subsequent simulations made on larger lattices, which were done on a Pentium II processor with 256 MB RAM working under the UNIX FreeBSD operating system. The code is given in Appendix C.

Following constant values of the main simulation parameters were used, as in [31]

- mean system density $\rho_m = 3.5$
- conventional temperature T = 0.550
- relaxation time $\tau = 0.8$

To investigate the shape of a sessile drop on a horizontal surface subjected to a gravitational field, we used a constant value of the parameter $\kappa=0.01$ of the LB model, which defines the surface tension, as discussed in [31]. We considered the two different cases, when the liquid is wetting the horizontal surface or not. For the first case, we adopted the value

$$\frac{\partial \mu}{\partial x_n} = 0.01 \tag{3.1}$$

of the normal derivative of the chemical potential on the lower wall surface. This value of the derivative accounts for the wetting interaction between the liquid drop and the wall. The second case is characterized by

$$\frac{\partial \mu}{\partial x_n} = -0.01 \tag{3.2}$$

which accounts for a non - wetting wall.

Figure 3.1 shows the time evolution of a sessile drop which wets the bottom wall and is subjected to an acceleration directed downwards (g = -0.00001). The drop spreads across the wall surface. Because the lattice was not large enough, the drop ends join laterally because of the periodic boundary conditions we used in the X direction, so that a flat liquid surface is recovered in the final state.

Figure 3.2 shows the equilibrium shape of a nonwetting drop subjected to different values of the gravitational acceleration. One can see that the drop becomes more and more flattened when the value of the gravitational acceleration increases.

3.2 Contact angle

We used a 1024×192 lattice. The necessary CPU time to perform 100,000 time steps was approx. 72 hours for each data set. Several values of the surface tension coefficient κ were considered, while the normal derivative of the chemical potential on the lower wall surface was $\frac{\partial \mu}{\partial x_n} = 0.01$. To account only for the effect of surface tension on the contact angle, no gravitational acceleration was considered ($\mathbf{g} = 0$).

Figures 3.3 - 3.6 show the time evolution of the lattice state for different values of the surface tension coefficient κ . When the value of the surface coefficient is large enough ($\kappa=0.01$), the drop spreads along the whole lattice domain, so that the contact angle cannot be determined because of the periodic boundary conditions on the left and right margin of the lattice (Figure 3.3). For smaller values of the surface tension, coefficient, when the drop does not spread along the whole bottom wall, the contact angle α may be determined using the following procedure (Figure 3.7), which takes into account the fact that the upper drop border becomes a circular arc when the equilibrium configuration is reached (i.,e., when the number of time steps is large enough). The drop height is denoted h, while its width is denoted w.



t = 0 time steps



 $t\,=\,1,000$ time steps



 $t\,=\,2,000$ time steps

Figure 3.1: Time evolution of the shape of a wetting drop.



t = 3,000 time steps



t = 4,000 time steps



t = 5,000 time steps

Figure 3.1: (continued) Time evolution of the shape of a wetting drop.



t = 6,000 time steps



t = 7,000 time steps



t = 10,000 time steps

Figure 3.1: (continued) Time evolution of the shape of a wetting drop.



t = 30,000 time steps

Figure 3.1: (continued) Time evolution of the shape of a wetting drop.

If R is the arc radius, we have

$$h = R(1 - \cos \alpha) \tag{3.3}$$

$$w = 2R\sin\alpha \tag{3.4}$$

$$\frac{h}{w} = \frac{1 - \cos \alpha}{2 \sin \alpha} = \frac{1}{2} \tan \frac{\alpha}{2} \tag{3.5}$$

$$\cos \alpha = 2 \arctan \left(\frac{2h}{w}\right)$$
 (3.6)

Table 3.1 shows the values of the contact angles for the cases shown in figures 3.4 - 3.6.

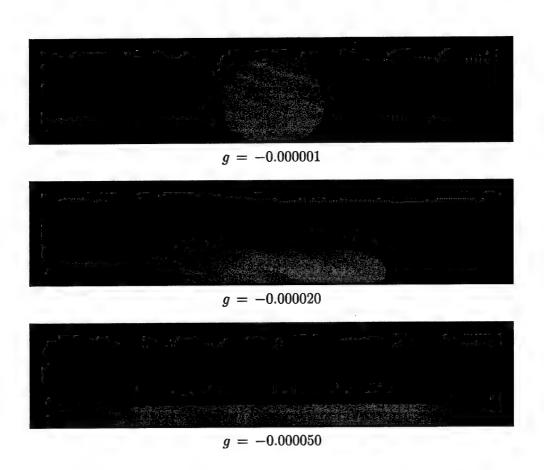


Figure 3.2: Shape of a non - wetting drop subjected to different values of the gravitational field.

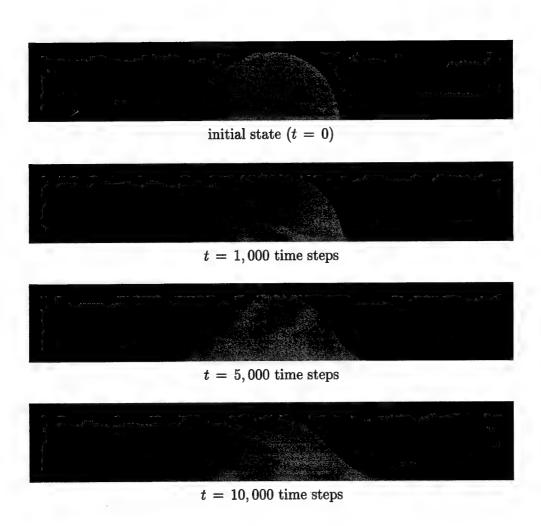


Figure 3.3: Drop shape during simulation with $\kappa = 0.0100$.

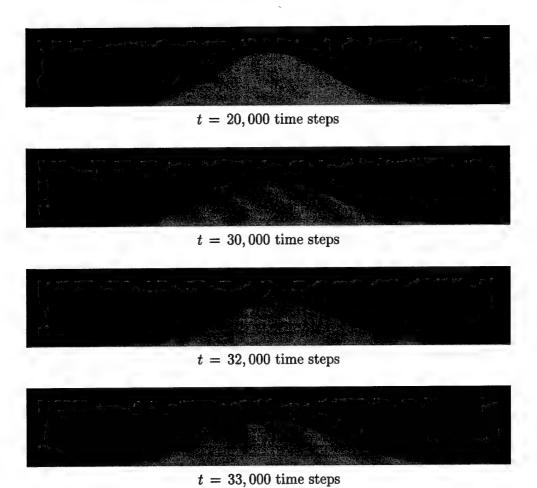


Figure 3.3: (continued) Drop shape during simulation with $\kappa = 0.0100$.

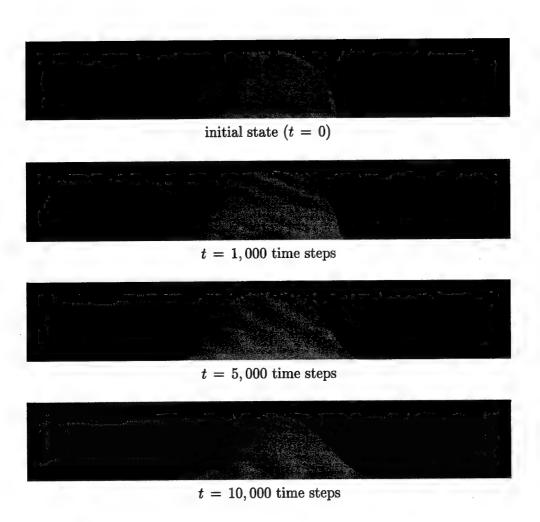


Figure 3.4: Drop shape during simulation with $\kappa = 0.0094$.

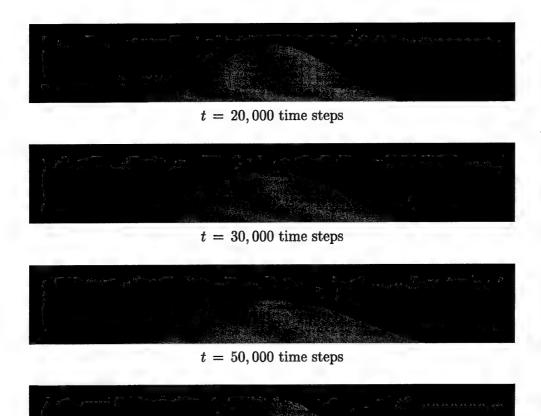


Figure 3.4: (continued) Drop shape during simulation with $\kappa=0.0094$.

t = 100,000 time steps

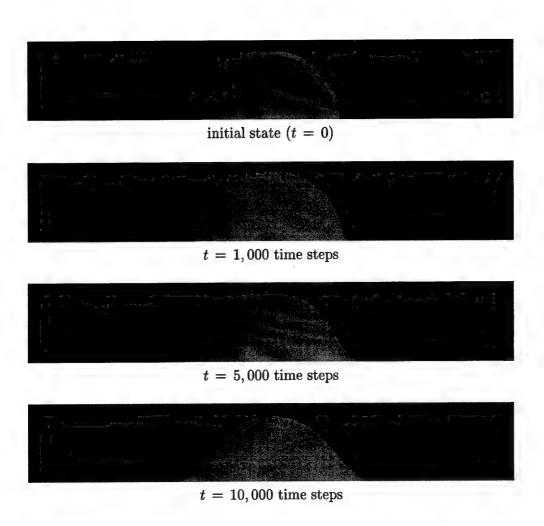


Figure 3.5: Drop shape during simulation with $\kappa = 0.0090$.

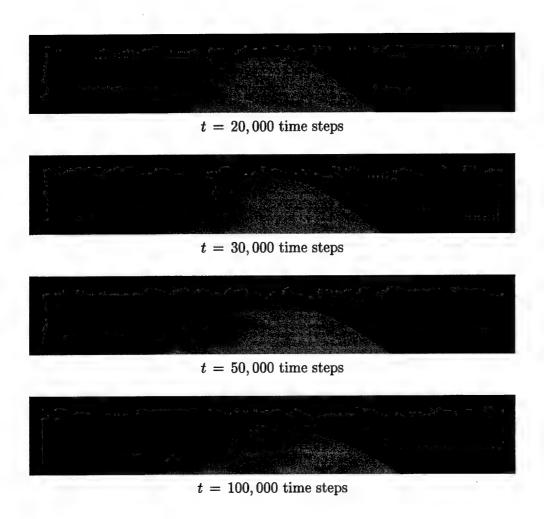


Figure 3.5: (continued) Drop shape during simulation with $\kappa=0.0090$.

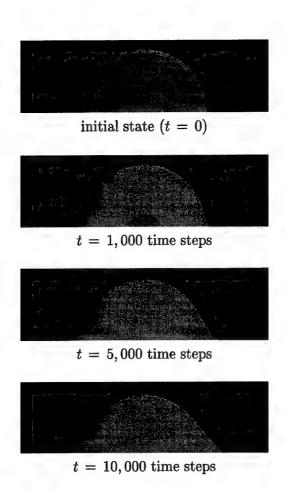


Figure 3.6: Drop shape during simulation with $\kappa = 0.0088$.

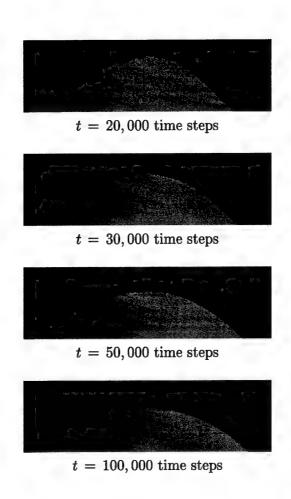


Figure 3.6: (continued) Drop shape during simulation with $\kappa=0.0088$.

Table 3.1: Influence of the surface tension coefficient κ on the contact angle.

κ	$\cos \alpha$
0.0094	0.9097
0.0090	0.7625
0.0088	0.6671

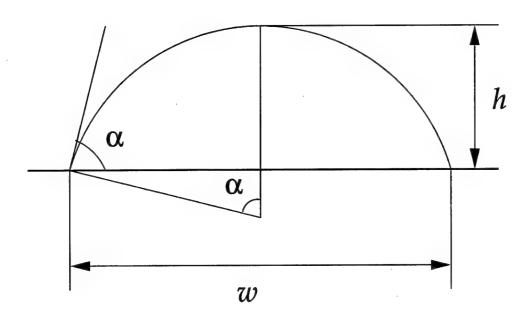


Figure 3.7: Determination of the value of the contact angle α .

References

- V. SOFONEA, Lattice Boltzmann Models for Multicomponent Fluids, Intermediate Report #002, Contract SPC - 98 - 4061 No. F61775 - 98 - WE101, EOARD (January 1999).
- [2] W. H. PRESS, S. A. TEUKOLSKY, W. T. Vetterling, B. P. FLAN-NERY, Numerical Recipes in Fortran: The Art of Scientific Computing, Cambridge University Press, Cambridge, 1992.
- [3] C. HIRSCH, Numerical Computation of Internal and External Flows, volume I: Fundamentals of Numerical Discretization, John Wiley and Sons, Chichester, New York, 1988.
- [4] K. W. MORTON, D. F. MAYERS, Numerical Solutions of Partial Differential Equations, Cambridge University Press, Cambridge, 1996.
- [5] X. HE, L. S. LUO, M. DEMBO, Some Progress in Lattice Boltzmann Method. Part I. Nonuniform Mesh Grids, Journal of Computational Physics 129 (1996) 357.
- [6] X. HE, G. D. DOOLEN, Lattice Boltzmann method on a curvilinear coordinate system: Vortex shedding behind a circular cylinder, Physical Review E56 (1997) 434.
- [7] X. HE, L. S. LUO, M. DEMBO, Some Progress in Lattice Boltzmann Method: Reynolds number enhancement in simulations, Physica A 239 (1997) 276.
- [8] X. HE, Error analysis for the interpolation supplemented lattice Boltzmann equation scheme, International Journal of Modern Physics C8 (1997) 737.

- [9] K. H. HUEBNER, The Finite Element Method for Engineers, John Wiley and Sons, New York, 1975.
- [10] X. Y. HE, L. S. LUO, Lattice Boltzmann model for the incompressible Navier - Stokes Equation, Journal of Statistical Physics 88 (1997) 927.
- [11] D. H. ROTHMAN, S. ZALESKI, Lattice Gas Cellular Automata, Simple Models of Complex Hydrodynamics, Cambridge University Press, Cambridge, 1997.
- [12] P. S. PERERA, R. F. SEKERKA, Nonsolenoidal flow in a liquid diffusion couple, Physica of FLuids bf9 (1997) 376.
- [13] X. SHAN, D. DOOLEN, Diffusion in a multicomponent lattice Boltzmann equation model, Physical Review E54 (1996) 3614.
- [14] X. SHAN, H. CHEN, Lattice Boltzmann Model for Simulating Flows with Multiple Phases and Components, Physical Review E47 (1993) 1815.
- [15] V. SOFONEA, Lattice Boltzmann Approach to Collective Particle Interactions in Magnetic Fluids, Europhysics Letters 25 (1994) 385.
- [16] V. SOFONEA, Lattice Boltzmann Model for Magnetic Fluids, Romanian Reports in Physics 47 (1995) 307.
- [17] X. SHAN, G. DOOLEN, Multicomponent Lattice Boltzmann model with interparticle interaction, Journal of Statistical Physics 81 (1995) 379.
- [18] B. CHOPARD, M. DROZ, Cellular Automata Modeling of Physical Systems, Cambridge University Press, Cambridge, 1998.
- [19] R. F. SEKERKA (Principal Investigator): Grant NAG3 1875, Fluid Physics in a Stochastic Acceleration Environment: Diffusion and Non – Solenoidal Flows, Carnegie Mellon University, Pittsburgh, Pennsylvania; Fluid Physics in a Stochastic Acceleration Environment: A Lattice Boltzmann Approach (Progress Report, Carnegie Mellon University, February 1999).

- [20] X. HE, L. S. LUO, Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation, Physical Review E56 (1997) 6811.
- [21] X. SHAN, X. HE, Discretization of the Velocity Space in the Solution of the Boltzmann Equation, Physical Review Letters 80 (1998) 65.
- [22] X. HE, X. SHAN, G. D. DOOLEN, Discrete Boltzmann equation model for nonideal gases, Physical Review E57 (1998) R13.
- [23] D. ROTHMAN, S. ZALESKI, Lattice gas models of phase separation: interfaces, phase transitions and multiphase flow, Reviews of Modern Physics 66 (1994) 1417.
- [24] M. R. SWIFT, W. R. OSBORN, J. M. YEOMANS, Lattice Boltzmann Simulation of Non-Ideal Fluids, Physical Review Letters 75 (1995) 830.
- [25] W. R. OSBORN, E. ORLANDINI, M. R. SWIFT, J. M. YEOMANS, J. R. BANAVAR, Lattice Boltzmann Study of Hydrodynamic Spinodal Decomposition, Physical Review Letters 75 (1995) 4031.
- [26] E. ORLANDINI, M. R. SWIFT, J. M. YEOMANS, Lattice Boltzmann Model of Binary Fluid Mixtures, Europhysics Letters 32 (1995) 463.
- [27] M. SWIFT, E. ORLANDINI, W. R. OSBORN, J. YEOMANS, Lattice Boltzmann simulations of liquid - gas and binary fluid systems, Physical Review E54 (1996) 5041.
- [28] G. GONNELLA, E. ORLANDINI, J. M. YEOMANS, Lattice Boltzmann simulations of lamellar and droplet phases, Physical Review E58 (1998) 480.
- [29] K. R. MECKE, V. SOFONEA, Morphology of spinodal decomposition, Physical Review **E56** (1997) R3761 – 3764.
- [30] V. SOFONEA, K. R. MECKE, Morphological Characterization of Spinodal Decomposition Kinetics, European Physical Journal B8 (1999) 99 – 112.
- [31] V. SOFONEA, Lattice Boltzmann Models for Multicomponent Fluids, Preliminary Report #001, Contract SPC - 98 - 4061 No. F61775 - 98 - WE101, EOARD (August 1998).

Appendix A wet9 code

wet9head.h

Jun 26 1999 18:38

<pre> wet9head.h definition of global variables const double woll (double) 3) ((double) 2); three_over_two = ((double) 3) ((double) 2); double wawall_top, "a_uywall_top, wawall_bot; double *a_uxwall_top, "a_uywall_left, wawall_bot; int a_nnodes_x, na_nnodes_y, a_unodes_all, lambda; int a_nnodes_x, na_nnodes_y, *a_unodes_all, lambda; int *a_key_init, *a_key_boundary, *a_key_scheme, *a_key_interface; int *a_noycles, *a_niter_cycle, witer_init; iter, miter; double engal, capeed, sepecd2, caforce_y; double thin, inc, inc, inc, inc, inc, inc, inc, i</pre>	Jun 26 1999 18:38 wet9head.h Page 1
wet9head.h — definition of global variables ***********************************	在 化苯基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲基甲
cdef MAIN_HEADER const double kboltz=1.381e-23, amu=1.661e-27, temp=300.0 kboltz = J/(molecules*K), amu = (kg), temp = (K); */ const double w0 = ((double) 1) / ((double) 36); w1 = ((double) 1) / ((double) 36); three_over_two = ((double) 3) / ((double) 2) const double three = ((double) 3), three_over_two = ((double) 3) / ((double) 2) double uxwall_left, uywall_left, uxwall_right, uywall_ri double *a_uxwall_left, uywall_left, uxwall_right, uywall_ri th nnodes_x, nnodes_y, nnodes_all, lambda; int *a_nodes_x, nnodes_lu, *a_uxwall_lot, *a_lambda; int *a_nodes_x, a_nnodes_y, *a_key_scheme, *a_key_lint *a_key_init, *a_key_boundary, *a_key_scheme, *a_key_lint *a_niter_cycle, niter_init, iter, niter; int *a_noycles, *a_niter_cycle, niter_init, iter, niter; int *a_noycles, *a_niter_cycle, niter_init, iter, niter; double length_x, length_y, delta_x, delta_y, delta_t, ct cspeedl, cspeedl, cspeedl2, cspeedl2, csforce_y; double length_x, length_y, delta_x, delta_y, delta_t, ct cspeedl, cspeedl, cspeedl2, cspeedl2, csforce_y; double timinus, tleenter, tlplus; double timinus, tleenter, timplus, tout, tout, tout, tout, tout, minusl, double tini, tini, tiniplus, tiniplus, tout, tout, tout, tout, double tini, tiniplus, tiniplus, tout, tout, tout, tout, inpulus, double tini, tiniplus, tiniplus, tout, tout, tout, tout, double tini, tiniplus, tiniplus, tout, tout, tout, tout, double tini, tiniplus, isenter! isplus, isminus!, iscenter; double isminus!, iscenter!, isplus; double isminus!, a_a_length_y, *a_delta_x, *a_delta_y, *a_delta_x, *a_delta_y, *a_delta_x, *a_delta_y, *a_delta	head.h definition of global variables ************************************
<pre>const double kboltz=1.381e-23, amu=1.661e-27, temp=300.0 kboltz = J/(molecules*K), amu = (kg), temp = (K); */ const double w0 = ((double) 9), w1 = ((double) 1) / ((double) 9), w2 = ((double) 1) / ((double) 36); const double three = ((double) 3), const double three_over_two = ((double) 3) / ((double) 2) inie_over_two = ((double) 9) / ((double) 2) double uxwall_top, uywall_top, wauxwall_bot, uywall_right, int nondes_x, wa_uywall_left, uxwall_right, uywall_right, int nondes_x, nondes_y, nondes_all, lambda; int key_init, key_boundary, key_point, key_scheme, key_int key_init, key_boundary, ka_nter_init, iter, niter; int waim, ncycles, niter_cycle, niter_init, iter, niter; int nsim, ncycles, niter_cycle, niter_init, iter, niter; int xa_ncycles, a_niter_cycle, niter_init, iter, niter; int the nsim, ncycles, niter_cycle, ra_niter_init, iter, niter; int the nsim, ncycles, niter_cycle, ra_niter_init, iter, niter; double length_x, length_y, delta_x, delta_y, delta_t, cfcree_y; double comminus; oneminus; oneplus; oneplus; double timinus, tlcenter, tlplus; double timinus, tlciplus; tiniplus; toutiminus, double timinus, tlcenter, isplus, tout, toutiminus, double ini, ini, tiniplus; tiniplus; toutiminus, double isminus!, iscenter; isplus, double a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_force_x, *a_length_y, *a_delte_x, *a_delte</pre>	*ifdef MAIN_HEADER
<pre>kboltz = J/(molecules*K), amu = (kg), temp = (K); */ const double w0 = ((double) 1) / ((double) 9), w1 = ((double) 1) / ((double) 9), w2 = ((double) 1) / ((double) 36); const double three = ((double) 3) / ((double) 2) nine_over_two = ((double) 9) / ((double) 2) double uxwall_top, uywall_top, uxwall_bot, uywall_bot; double uxwall_left, uywall_left, uxwall_tight, uywall_right, int mnodes_x, nnodes_y, mnodes_all, lambda; int *a_nnodes_x, nnodes_y, wa_nnodes_all, *a_lambda; int *a_nnodes_x, nnodes_y, wa_nnodes_all, wa_lambda; int *a_nnodes_x, representation, terrinit, iter, niter; int *a_nnodes_x, representation, wa_key_scheme, key_int nsim, ncycles, niter_cycle, niter_init, iter, niter; int *a_ntycles, *a_niter_cycle, ra_niter_init, double length_x, length_y, delta_x, delta_y, delta_t, ct cspeedl, cspeedl, cspeedl2, cspeedl2, cspeedl2, cfll, cfl2, force_x, force_y, csforce_x, csforce_y; double timinus, tlcenter, tlplus; double timinusl, tlfplusl, timinusl, infplus2, toutl, toutlminusl, double timi, tiniplus1, tiniplus2, toutl, toutlminusl, double timi, tiniplus1, tiniplus2, toutl, toutlminusl, double timinusl, iscenteri, isplus1, isminus2, iscenteri, double isminus1, iscenteri, isplus1, isminus2, iscenteri, wa_force_x, *a_length_y, *a_delta_x, *a_delta_y, *a_force_x, *a_length_y, *a_delta_x, *a_delta_y, *a_force_x, *a_length_y, *a_delta_y, *a_force_x, *a_length_y, *a_delta_y, *a_force_x, *a_length_y, *a_delta_y, *a_delta_</pre>	amu=1.661e-27,
<pre>ouble w0 = ((double) 9), w1 = ((double) 1) / ((double) 36); w2 = ((double) 1) / ((double) 36); w2 = ((double) 1) / ((double) 36); cuble three = ((double) 3), three_over_two = ((double) 3) / ((double) 2) nine_over_two = ((double) 3) / ((double) 2) nine_over_two = ((double) 3) / ((double) 2) uxwall_left, uywall_left, uxwall_right, uywall_top, *a_uxwall_bot, uywall_ti *a_uxwall_left, va_uywall_left, *a_uxwall_bot, *a_uxwall_bot, *a_uxwall_bot, *a_uxwall_bot, *a_uxwall_bot, *a_uxwall_left, *a_left boundary, key_boundary, *a_key_scheme, *a_key_scheme, *a_key_int, *a_key_boundary, *a_niter_init, iter, niter; noycles, *a_niter_cycle, *a_niter_init, iter, niter; length_x, length_y, delta_x, delta_t, delta_t, ctrongenter, torce_x, corce_y, csforce_y, csforce_y, csforce_y, csforce_y, csforce_y, csforce_y, linthus, tinplus, tinhus, tinplus, tinhus, tinplus, tinhus, tinplus, tinhus, tinplus, t</pre>	kboltz = $J/$ (molecules*K), amu = (kg), temp = (K);
<pre>couble three = ((double) 3),</pre>	<pre>double w0 = ((double) 4) / ((double) w1 = ((double) 1) / ((double) w2 = ((double) 1) / ((double)</pre>
uxwall_top, uywall_top, uxwall_bot, uywall_ri *a_uxwall_left, uywall_left, uxwall_right, uywall_ri *a_uxwall_left, *a_uywall_left, *a_uxwall_right, des_x, nnodes_y, nnodes_all, lambda; innodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda; innt, key_boundary, key_point, key_scheme, key_l key_init, *a_key_boundary, *a_key_scheme, *a_key_ inter_cycle, niter_init, iter, niter; ncycles, *a_niter_cycle, *a_niter_init, incycles, 'a_niter_cycle, *a_niter_init, copeedi, cspeedi2, cspeedi2, cfll, cfl2, force_x, force_y, csforce_x, csforce_y; gforce, kforce, kbound1, kbound2; oneminus1, oneminus2, oneplus1, oneplus2; tlminus, tlcenter, tiplus; tlminus, tlcenter, tiplus; tlminus, tlcenter, tiplus; tlminus1, tinplus1, tinplus2, inlplus2; tin, tinplus1, tinplus2, tout1, toutlminus1, tin2, tinplus1, tinplus2, tout2, tout2minus1, tin2, tinplus1, tinplus2, tout2, tin3, tinplus1, tinplus2, *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_length_x, *a_force_y, *a_gforce; mass1, mass2, tau1, tau2;	<pre>double three = ((double) 3), three_over_two = ((double) 3) / ((double) 2) nine_over_two = ((double) 9) / ((double) 2)</pre>
uxwall_left, uywall_left, uxwall_right, uywall_ri *a_uxwall_top, *a_uywall_top, *a_uxwall_bot, *a_u *a_uxwall_left, *a_uywall_top, *a_uxwall_bot, *a_u *a_uxwall_left, *a_uywall_left, *a_uxwall_right, des_x, nnodes_y, nnodes_all, lambda; .init, key_boundary, key_point, key_scheme, key_i .init, key_boundary, key_point, key_scheme, *a_keyinit, key_boundary, key_point, iter, niter; .init, *a_key_boundary, *a_key_scheme, *a_keyin, ncycles, *a_niter_cycle, niter_init, iter, niter; .incycles, *a_niter_cycle, *a_niter_init, iter, niter; .incycles, *a_niter_cycle, *a_niter_init, iter, niter; .incycles, *a_niter_cycle, *a_niter_init, cfl2; .gopeed, cspeed2, cspeed2, cspeed2, cfl1, cfl2; .gopeed, cspeed2, cspeed2, cspeed2, cfl1, cfl2; .cspeed, cspeed3, cspeed2, cspeed2, cfl1, cfl2; .cspeedd, cspeed3, cspeed2, cspeed2, csforce_y; .tinnus, tlcenter, tlplus; .tinnus, tlcenter, tlplus; .tinnus, tlcenter, tlplus; .tinnus, tlplus1, innzplus1, innplus2; .tin, tinnplus1, tinnplus2, tout; .tinnus1, tinplus1, tinnplus2, tout; .tinnus1, tinplus1, tinnplus2, tout; .tinnus1, tinnplus1, tinnplus2, tout; .tinnplus1, tinnplus1, tinnplus2, toutiminus1, .tinnplus1, tinnplus2, toutiminus2, .tinnt, tinnplus1, tinnplus2, toutiminus1, .tinnplus1, tinnplus2, toutiminus2, .tinn, tinnplus1, tinnplus2, toutiminus1, .tinnplus1, tinnplus2, toutiminus2, .tinnt, tinnplus1, tinnplus2, .tinnts2, toutiminus1, .tinnts2, toutiminus1, .tinnts2, toutiminus2, .tinnts2, .tinnts2, toutiminus2, .tinnts2, .tin	uxwall_top, uywall_top, uxwall_bot,
*a_uxwall_top, *a_uywall_top, *a_uxwall_bot, *a_uxwall_left, *a_uxwall_left, *a_uxwall_left, *a_uxwall_right, des_x, nnodes_y, nnodes_all, lambda; nnodes_x, *a_nnodes_all, lambda; nnodes_x, *a_nnodes_all, *a_lambda; nnodes_x, *a_nnodes_all, *a_lambda; lant, key_boundary, key_point, key_scheme, key_lev_init, *a_key_boundary, *a_key_scheme, *a_key_ncycles, niter_cycle, niter_init, iter, niter; ncycles, a_niter_cycle, a_niter_init, iter, niter; ncycles, ranter_cycle, *a_niter_init; force_y, force_y, cspeed2, cfil, cfl2, gforce, kforce, kbound1, kbound2; force_y; force_y, cscore_y; framinus, tlcenter, tiplus; nneplus2; tlfminus, tlcenter, tiplus1, tinfplus2, tinfplus2, tout1, tinplus1, tinfplus1, tinfplus2, tout1, tinplus1, tinplus2, tout1, toutIminus1, tinfplus1, tinplus2, tout2, tout2, tout2minus1, tinfplus1, tinplus1, tinplus1, tinplus1, tinplus2, tout2, tout2minus1, tinfplus1, tinplus2, tout1, tinplus1, tinplus2, tout2, tout2minus1, tinfplus1, tinplus2, tout2, tout2minus1, tinfplus1, tinplus2, tout2, tout2minus1, tinplus1, tinplus2, tout2, tout2minus1, tinfplus1, tinplus2, tout2, tout2minus1, tinfplus2, tout2, tout2, tout2minus1, tout2, tout2, tout2, tout2, tout2	uxwall_left, uywall_left, uxwall_right,
*a_uxwall_left, *a_uywall_left, *a_uxwall_right, des_x, nnodes_y, nnodes_all, lambda; nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda; _init, key_boundary, key_point, key_scheme, key_lev_init, *a_key_boundary, *a_key_scheme, *a_key_noint, iter, niter; ncycles, *a_niter_cycle, *a_niter_init; length_x, length_y, dalta_x, delta_y, delta_t, ct speedi, cspeedi, tininus, tiniplus; tiniplu	*a_uxwall_top, *a_uywall_top, *a_uxwall_bot, *a_u
des_x, nnodes_y, nnodes_all, lambda; nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda; _init, key_boundary, key_point, key_scheme, key_interface; _init, key_boundary, *a_key_scheme, *a_key_interface; key_init, *a_key_boundary, *a_key_scheme, *a_key_interface; .m, ncycles, *a_niter_cycle, niter_init, iter, niter; ncycles, *a_niter_cycle, *a_niter_init; length_x, length_y, delta_x, delta_y, delta_t, ctaul, ctaul capeedi, cspeedi, cspee	*a_uxwall_left, *a_uywall_left, *a_uxwall_right,
nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda; .init, key_boundary, key_point, key_scheme, key_interface; .key_init, *a_key_boundary, *a_key_scheme, *a_key_interface; .m, ncycles, *a_niter_cycle, niter_init, iter, niter; .ncycles, *a_niter_cycle, *a_niter_init, length_x, length_y, delta_x, delta_y, delta_t, ctaul, ctaul capeadl, capeadl2, capeadl2, capeadl2, capeadl2, caforce_y; force_y, csforce_y, csforce_y; .csforce_y; .csforce_y; .csforce_y; .csforce_y; .toneminus1, oneminus2, oneplus1; .timinus, ticenter, tiplus; .timinus, ticenter, tiplus; .timinus, tiplus1, tiffius1, tiffius1, tiffius1, tiffius1, tiffius1, tiffius1, tiffius2, tiffinus2, tout1, toutlaminus1, .tinlplus1, tinlplus2, tout1, toutlaminus1, toutlaminus1, .tinlplus1, tinlplus2, tout1, toutlaminus1, toutlaminus1, tinlplus1, tinlplus2, tout2, tout2, tout2, tout2, tout2, tout2, tout2, tout2, tout2, isplus2, .a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_force_x, *a_force	nnodes_x, nnodes_y, nnodes_all,
Linit, key_boundary, key_point, key_scheme, key_interface; key_init, *a_key_boundary, *a_key_scheme, *a_key_interface; .m, ncycles, inter_cycle, niter_init, iter, niter; ncycles, *a_niter_cycle, rainter_init; length_x, length_y, delta_x, delta_y, delta_t, ctaul, ctauc capeed1, capeed2, cspeed22, cfl1, cfl2, force_x, force_y, csforce_x, csforce_y; gforce_x, force_y, csforce_x, csforce_y; tfminus, loemtinus2, neplus1, noplus2; tfminus, tlcenter, tlplus; tfminus, tlcenter, tlplus; tfminus, tlfplus1, tlfminus2, tlfplus2; tlfminus1, tlfplus1, tlfminus2, toutlinus2, upAminus2, tln1, tinplus1, tln2plus2, tout1, toutlminus1, toutlminus2, tin1, tinplus1, tln2plus2, tout1, toutlminus1, toutlminus2, isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; sminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; mass1, mass2, tau1, tau2;	*a_nnodes_x, *a_nnodes_y, *a_nnodes_all,
key_init, *a_key_boundary, *a_key_scheme, *a_key_interface; m, ncycles, niter_cycle, niter_init, iter, niter; ncycles, *a_niter_cycle, *a_niter_init; length_x, length_y, delta_x, delta_t, delta_t, ctaul, length_x, length_y, csforce_x, csforce_y; force_x, force_y, csforce_x, csforce_y; gforce, kforce, kbound1, kbound2; oneminus1, oneminus2, oneplus1, oneplus2; timinus, tlcenter, tpplus; tlminus, tlcenter, tpplus; tlminus, tlcenter, tpplus; tlminus, tlcenter, tpplus; tlminus1, tlfplus1, tlfminus2, tlfplus2; in1, in2, in1plus1, in2plus1, in1plus2, in2plus2; up1, up2, uplminus1, up2minus1, uplminus2, uplminus2, tin1, tin1plus1, tin1plus2, tout1, toutlminus2, tin1, tin1plus1, tin1plus2, tout2, tout1zminus1, toutlminus2, isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_* *a_force_x, *a_force_y, *a_gforce; mass1, mass2, tau1, tau2;	key_init, key_boundary, key_point, key_scheme,
m, ncycles, niter_cycle, niter_init, iter, niter; ncycles, *a_niter_cycle, *a_niter_init; length_x, length_y, delta_x, delta_t, delta_t, cful, cspeed1, cspeed2, cspeed2, cspeed2, cful, cfl2, force_x, force_y, csforce_x, csforce_y; gforce, kforce, kbound1, kbound2; oneminus1, oneminus2, oneplus1, oneplus2; timinus, t2center, t1plus; timinus, t2center, t2plus; t1fminus, t2center, t2plus; t1fminus, t2center, t2plus; t1fminus, t2center, t3plus; t1fminus1, t1fplus1, t1fminus2, t1fplus2, updinus2, tini2plus1, tiniplus2, tout2, tout2minus1, tout1minus2, tini2plus1, tiniplus2, tout2, tout2minus1, tout1minus2, tini2plus1, tiniplus2, isminus1, iscenter1, isplus1, isminus1, iscenter1, isplus2, isminus2, iscenter2, isplus2; mass1, mass2, tau1, tau2;	*a_key_init, *a_key_boundary, *a_key_scheme,
ncycles, *a_niter_cycle, *a_niter_init; length_x, length_y, delta_x, delta_t, delta_t, ctaul, ctaul cspeedl, cspeedl2, cspeedl2, cspeedl2, force_x, force_y, csforce_x, csforce_y; gforce, kforce, kbound1, kbound2; oneminus1, oneminus2, oneplus1, oneplus2; tlminus, tlcenter, tlplus; tlminus, tlcenter, tlplus; tlminus, tlcenter, tlplus; tlfminus1, tlfplus1, tlfminus2, tlfplus2; in1, in2, in1plus1, in2plus1, in1plus2, upZminus2, upZminus1, toutlminus2, tin1, tin1plus1, tlinplus2, tout1, toutlminus1, toutlminus2, tin1, tin1plus1, tin1plus2, tout1, toutlminus1, toutlminus2, tin1, tin1plus1, tin1plus2, tout2, tout2minus1, toutZminus2, isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; mass1, mass2, tau1, tau2;	nsim, ncycles, niter_cycle, niter_init, iter,
length_x, length_y, delta_x, delta_y, delta_t, ctaul, craus capeed1, capeed12, capeed12, capeed12, cfl1, cfl1, cfl2, force_y, csforce_y; csforce_y; lift, cfl1, cfl2, gforce, x force_y, csforce_y; lift, cfl1, cfl1, cfl2, coneminus1, oneminus2, oneplus1, oneplus2; tlminus, tlcenter, tlplus; tlminus, tlcenter, tlplus; tlminus2, tlfplus2; inlplus1, tlfplus1, tlfplus1, tlfplus1, tlfplus1, tlfplus1, tlfplus1, tuntlminus2, upminus2, upminus2, upminus1, upc2, uplminus1, tlplus1, tlnlplus2, tout1, toutlminus1, tuntlminus1, tlnlplus1, tlnlplus2, tout1, toutlminus1, toutlminus2, incenter1, isplus1, isminus2, iscenter2, isplus2; sminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2, *a_length_x, *a_delta_x, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_force_x, *a_force_y, *a_delta_x, *a_force_x, *a_fo	*a_ncycles, *a_niter_cycle,
<pre>gforce, kforce, kbound1, kbound2; oneminus1, oneminus2, oneplus1, oneplus2; tIminus, tlcenter, tIplus; tZminus1, tlcenter, t2plus; tlfminus1, tlfplus1, tlfminus2, tlfplus2; in1, in2, in1plus1, in2plus1, in1plus2, in2plus2; up1, up2, up1minus1, up2minus1, up1minus2, up2minus2, up1minus2, up2minus1, tin1plus2, tout1, tin1plus1, tin1plus2, tout2, tout1, tin1plus1, tin2plus2, tout2, tout2minus1, iscenter1, isplus1, isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; *a_length_x, *a_length_y, *a_delta_x, *a_delta_x, *a_force_x, *a_force_y, *a_gforce; mass1, mass2, tau1, tau2;</pre>	<pre>length_x, length_y, delta_x, delta_y, delta_t, ctaul, cspeedl, cspeed2, cspeed12, cspeed22, cf11, cf12, force_x, force_y, csforce_x, csforce_y;</pre>
oneminus, cneminus2, oneplus1, oneplus2; tlminus, tlcenter, tlplus; tlminus, tlcenter, t2plus; tlfminus1, tlfplus1, tlfminus2, tlfplus2; in1, in2, in1plus1, in2plus1, in1plus2, in2plus2; up1, up2, up1minus1, up2minus1, upfminus2, up2minus1, tin1plus2, tout1, tin1plus1, tin1plus2, tout1, tout1minus1, tin2plus1, tin1plus2, tout2, tout2, tout2minus1, tout2minus2, isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2; *a_length_x, *a_length_y, *a_delta_x, *a_delta_x, *a_force_x, *a_force_y, *a_gforce; mass2, tau1, tau2;	gforce, kforce,
tlminus, tlcenter, tlplus; t2minus, t2center, t2plus; t1fminus1, t1fplus1, t1fminus2, t1fplus2; in1, in2, in1plus1, in2plus1, in1plus2, in2plus2; tin1, in2, uplminus1, up2minus1, up1minus2, up1minus2, up1minus1, tin1plus2, tout1, tout1minus1, tout1minus1, tin2plus1, tin1plus2, tout2, tout2minus1, tout2minus1, isplus1, isplus2, iscenter2, isplus2; *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_x, *a_force_x, *a_force_y, *a_force; mass2, tau1, tau2;	oneminus1, oneminus2, oneplus1,
inl, in2, inlplus1, in2plus1, inlplus2, in2plus2; up1, up2, up1minus1, up2minus1, up1minus2, up2minus2, up2minus1, up1minus2, up2minus1, tin1plus1, tin1plus2, tout1, toutlminus2, tin2, tin2plus1, tin2plus2, tout2, tout2minus1, tout2minus1, iscenter1, isplus1, isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2, *a_length_x, *a_length_y, *a_delta_x, *a_force_x, *a_force_y, *a_gforce; mass1, mass2, tau1, tau2;	tlminus, tlcenter, tlplus; t2minus, t2center, t2plus; t1fminus1, t1fplus1, t1fminus2, t1fplus2
*a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_ *a_force_x, *a_force_y, *a_gforce; mass1, mass2, tau1, tau2;	
massl, mass2, taul,	*a_length_x, *a_length_y, *a_delte_x, *a_delta_y, *a_delta_ *a_force_x, *a_force_y, *a_gforce;
	massl, mass2, taul,
double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2;	*a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1,
double nzerol, nzero2, nzerolleft, nzerolright, nzero2left, nzero2right;	nzerol, nzero2, nzerolleft, nzerolright, nzero2left,
<pre>double *a_nzerol, *a_nzerol, *a_nzerolleft, *a_nzerolleft,</pre>	*a_nzerol, *a_nzero2, *a_nzerolleft, *a_nzerolright, *a_nzeroZright;

<pre>char input_name[] = "wet9.input", output_name[] = "wet9.output";</pre>
char id_name[128], rez_name[128], xv_name[128];
<pre>double ww[9], ecx[9], ecy[9], ecx1[9], ecy1[9], ecx2[9], ecy2[9];</pre>
double nc10[9], nc11[9], nc12[9], nc13[9], nc14[9], nc15[9], nc17[9], nc17[9], nc17[9],
double nc20[9], nc21[9], nc22[9], nc23[9], nc24[9], nc25[9], nc26[9], nc27[9], nc28[9];
<pre>double ncbct10[9], ncbct11[9], ncbct12[9], ncbct13[9], ncbct14[9], ncbct15[9], ncbct17[9], ncbct17[9], ncbct18[9];</pre>
<pre>double ncbot20[9], ncbot21[9], ncbot22[9], ncbot23[9], ncbot24[9], ncbot25[9], ncbot26[9], ncbot27[9], ncbot28[9];</pre>
<pre>double nctop10[9], nctop11[9], nctop12[9], nctop13[9], nctop14[9], nctop15[9], nctop16[9], nctop17[9], nctop18[9];</pre>
<pre>double nctop20[9], nctop21[9], nctop22[9], nctop23[9], nctop24[9], nctop25[9], nctop26[9], nctop27[9], nctop28[9];</pre>
<pre>double ncleft10[9], ncleft11[9], ncleft12[9], ncleft13[9], ncleft15[9], ncleft16[9], ncleft17[9], ncleft18[9];</pre>
<pre>double ncleft20[9], ncleft21[9], ncleft22[9], ncleft23[9],</pre>
<pre>double ncright10[9], ncright11[9], ncright12[9], ncright13[9], ncright15[9], ncright15[9], ncright16[9], ncright17[9], ncright18[9];</pre>
<pre>double ncright20[9], ncright21[9], ncright22[9], ncright23[9], ncright24[9],</pre>
<pre>double null(9), null(9), nul2(9), nul3(9), nul4(9), nul5(9), nul7(9), nul7(9), nul8(9);</pre>
double nu20[9], nu21[9], nu22[9], nu23[9], nu24[9], nu25[9], nu26[9], nu27[9], nu28[9];
<pre>double nsitl[9], nsitr[9], nsibl[9], nsibr[9], ns2tl[9], ns2tr[9], ns2tr[9];</pre>
<pre>double nsbit1[9], nsbitx[9], nsbibx[9], nsb2tx[9], nsb2tx[9], nsb2tx[9], nsb2tx[9], nsb2tx[9],</pre>
<pre>double nstiti[9], nstitr[9], nstibi[9], nst2br[9], nst2t1[9], nst2tr[9], nst2br[9];</pre>
<pre>double nslitt[9], nslitr[9], nslibl[9], nsl2br[9], nsl2tl[9], nsl2tr[9], nsl2bl[9], nsl2br[9];</pre>
<pre>double nsritt[9], nsritr[9], nsrib1[9], nsribr[9],</pre>
<pre>double nlltl[9], nlltr[9], nllbl[9], nllbr[9], nl2tr[9], nl2tr[9], nl2br[9];</pre>
<pre>double nlbiti[9], nlbitr[9], nlbibl[9], nlbibr[9], nlb2t1[9], nlb2tr[9], nlb2br[9], nlb2br[9];</pre>
<pre>double nltltl[9], nltltr[9], nltlbl[9], nltlbr[9], nlt2tl[9], nlt2tr[9], nlt2br[9], nlt2br[9];</pre>
<pre>double nllit1[9], nllitr[9], nllib1[9], nllibr[9], nll2t1[9], nll2tr[9], nll2b1[9], nll2br[9];</pre>

wet9head.h

Jun 26 1999 18:38

	2000
nlritr[9], nlribl[9],	
nlr2t1[9], nlr2tr[9], nlr2b1[9], nlr2br[9];	
<pre>double ecxforce1[9], ecyforce1[9], ecxforce2[9], ecyforce2[9], ecprod1[9], ecprod2[9];</pre>	
int *boundary_mode;	
double *f10, *f11, *f12, *f13, *f14, *f15, *f16, *f17, *f18;	
double *£20, *£21, *£22, *£23, *£24, *£25, *£26, *£27, *£28;	
double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16, *ff17, *ff18;	
double *ff20, *ff21, *ff22, *ff23, *ff24, *ff25, *ff26, *ff27, *ff28;	
double *neq10, *neq11, *neq12, *neq13, *neq14, *neq15, *neq16, *neq17, *neq18;	
<pre>double *neq20, *neq21, *neq22, *neq24, *neq25, *neq26,</pre>	
<pre>double *source10, *source11, *source12, *source13, *source14, *source15,</pre>	
<pre>double *source20, *source21, *source22, *source23, *source24, *source25,</pre>	
double *sf10, *sf11, *sf12, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;	
double *sf20, *sf21, *sf22, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;	
double *uxloc, *uyloc, *uxloc1, *uyloc1, *uxloc2, *uyloc2;	
double *nloc1, *nloc2, *colorfield, *gradcolor_x, *gradcolor_y;	
<pre>double *gradn1x, *gradn2x, *gradn1y, *gradn2y;</pre>	
int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;	
‡ ⊕1se	
extern const double kboltz, amu, temp;	
extern const double w0, w1, w2;	
extern const double three, three_over_two, nine_over_two;	
<pre>extern double uxwall_top, uywall_top, uxwall_bot, uywall_bot;</pre>	
<pre>extern double uxwall_left, uywall_left, uxwall_right, uywall_right;</pre>	
<pre>extern double *a_uxwall_top, *a_uywall_top, *a_uxwall_bot, *a_uywall_bot;</pre>	
<pre>extern double *a_uxwall_left, *a_uywall_left,</pre>	
extern int nnodes_x, nnodes_y, nnodes_all, lambda;	
extern int *a_nnodes_x, *a_nnodes_y, *a_nnodes_ail, *a_lambda;	
extern int key_init, key_boundary, key_point, key_scheme, key_interface;	
extern int *a_key_init, *a_key_boundary, *a_key_scheme, *a_key_interface;	
extern int nsim, ncycles, niter_cycle, niter_init, iter, niter;	

extern double length, x, length, y, delta, x, delta, y, delta, t, ctaul, ctau2, cspeed1, cspeed12, cspeed22, cfl1, cfl2, force, x, force, y, csforce, y, csforce, y; extern double gforce, kforce, kbound1, kbound2;
extern double oneminus1, oneminus2, oneplus1, oneplus2;
extern double timinus, ticenter, tiplus; extern double timinus, ticenter, tiplus; extern double tifminus; tifplus; tifminus; tifplus;
extern double in1, in2, in1plus1, in2plus1, in1plus2, in2plus2; extern double up1, up2, up1minus1, up2minus2, up2minus2; extern double tin1, tin1plus1, tin1plus2, tout1, tout1minus1, tout1minus1, extern double tin2, tin2plus1, tin2plus2, tout2, tout2minus1, tout2minus2; extern double isminus1, iscenter1, isplus1, isminus2, iscenter2, isplus2;
<pre>extern double *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_t,</pre>
extern double mass1, mass2, tau1, tau2;
extern double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2;
extern double nzerol, nzero2, nzerolleft, nzero2left, nzero1right, nzero2right;
<pre>extern double *a_nzero1, *a_nzero2, *a_nzerolleft, *a_nzero2left,</pre>
extern char input_name[], output_name[];
extern char id_name[], rez_name[], xv_name[];
extern double ww[], ecx[], ecy[], ecx1[], ecy1[], ecx2[], ecy2[];
<pre>extern double nc10[], nc11[], nc12[], nc13[], nc14[], nc15[], nc16[], nc17[], nc18[];</pre>
extern double nc20[], nc21[], nc22[], nc23[], nc24[], nc25[], nc26[], nc26[], nc27[], nc28[];
<pre>extern double ncbot10[9], ncbot11[9], ncbot12[9], ncbot13[9], ncbot14[9], ncbot15[9], ncbot16[9], ncbot16[9],</pre>
<pre>extern double ncbot20[9], ncbot22[9], ncbot22[9], ncbot23[9], ncbot28[9], ncbot26[9], ncbot26[9], ncbot27[9], ncbot28[9];</pre>
<pre>extern double nctop10[9], nctop11[9], nctop12[9], nctop13[9], nctop14[9], nctop15[9], nctop16[9], nctop17[9],</pre>
extern double nctop20[9], nctop21[9], nctop22[9], nctop23[9], nctop24[9], nctop25[9],
<pre>extern double ncleft10[9], ncleft11[9], ncleft12[9], ncleft13[9], ncleft14[9], ncleft15[9], ncleft16[9], ncleft17[9], ncleft18[9];</pre>
<pre>extern double ncleft20[9], ncleft21[9], ncleft22[9], ncleft23[9], ncleft24[9], ncleft25[9], ncleft26[9], ncleft27[9], ncleft28[9];</pre>
<pre>extern double ncright10[9], ncright12[9], ncright13[9],</pre>
<pre>extern double ncright20[9], ncright21[9], ncright22[9], ncright23[9],</pre>

2
=
듣
O
ஓ
Ĕ
=
땦
8
22
⊏
_
-
Ē
ö
2
Ψ.
Ü
<u>e</u>
Ë
fonea
Ë
fone
ted by sofone:
fone
ted by sofone:

wet9head.h

Jun 26 1999 18:38	wet9head.h	Page 5	Jun 26 1999 18:38
extern double nu10[], nu11[], nu1: nu16[], nu17[], nu1	nu12[], nu13[], nu14[], nu15[], nu18[];		extern double *nloo
<pre>extern double nu20[], nu21[], nu2 nu26[], nu27[], nu2</pre>	nu20[], nu21[], nu22[], nu23[], nu24[], nu25[], nu26[], nu27[], nu28[];		extern double "gra extern int *nv1, *;
<pre>extern double nsiti[], nsitr[], nsibi[],</pre>	sslb[], nslbr[], is2bl[], ns2br[];	· · · ·	#endif
<pre>extern double nsbltl[], nsbltr[], nsb2tr[], nsb2tr[</pre>	nsblbl[], nsblbr[], nsb2bl[], nsb2br[];		
<pre>extern double nstitt[], nstitt[], nstibi[],</pre>	nstlbi[], nstlbr[], nst2bl[], nst2br[];		
<pre>extern double nslitt[], nslitt[], nslibt[],</pre>	nslib1[], nslibr[],		
<pre>extern double nsrltl[], nsrltr[],</pre>	<pre>nsrltl[], nsrltr[], nsrlbl[], nsrlbr[], nsr2tl[], nsr2tr[], nsr2bl[], nsr2br[];</pre>		
<pre>extern double nilti[], niltr[], nilbi[], nilbr[],</pre>	lllb1[], nllbr[], nl2b1[], nl2br[];		
<pre>extern double nlb1t[], nlb1tr[], nlb1b1[], nlb2t1[], nlb2tr[], nlb2b1[],</pre>	nibibi[], nibibr[],		-
<pre>extern double nlt1t1[], nlt1tr[],</pre>	. nltlb[], nltlbr[], . nlt2bl[], nlt2br[];		
<pre>extern double nllltl[], nllltr[], nll2tr[],</pre>	nlltt[[], nlltt[], nllibl[], nllibt[], nlltt[[], nllttt[], nll2bt[], nll2bt[],		
<pre>extern double nlr1t1[], nlr1tr[],</pre>	nlriti[], nlritr[], nlribl[], nlribr[], nlr2t1[], nlr2tr[], nlr2b1[], nlr2br[];		
extern double ecxforcel[], ecyfor ecyrod2[<pre>ecxforce1[], ecyforce1[], ecxforce2[], ecyforce2[], ecprod1[], ecprod2[];</pre>		- 11
extern int *boundary_mode;			
extern double *f10, *f11, *f12, *	*f13, *f14, *f15, *f16, *f17, *f18;		
extern double *f20, *f21, *f22, *	*f23, *f24, *f25, *f26, *f27, *f28;		
extern double *ff10, *ff11, *ff12, *ff18;	2, *ff13, *ff14, *ff15, *ff16,		
extern double *ff20, *ff21, *ff22, *ff28;	2, *ff23, *ff24, *ff25, *ff26,	•	
extern double *neq10, *neq11, *ne *neq17, *neq18;	*neq12, *neq13, *neq14, *neq15, *neq16,		
extern double *neq20, *neq21, *ne *neq21, *neq28;	*neq22, *neq23, *neq24, *neq25, *neq26,		
<pre>extern double *source10, *source11,</pre>	11, *source12, *source13, *source14, 16, *source17, *source18;		
<pre>extern double *source20, *source21,</pre>	21, *source22, *source23, *source24, 26, *source27, *source28;		
extern double *sf10, *sf11, *sf12,	2, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;	-	
extern double *sf20, *sf21, *sf22,	2, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;		
extern double *nxloc. *uvloc. *ux	*uxloc1, *uyloc1, *uxloc2, *uyloc2; ;		

	extern double *nloc1, *nloc2, *colorfield, *gradcolor_x, *gradcolor_y;	
	extern double *gradnlx, *gradn2x, *gradnly, *gradn2y;	
	extern int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;	
	#endif	-
•		

/*************************************		- age -
* wetginout.c ************************************	***********	
#include <stdio.h> #include <stdiib.h> #include <string.h> #include <math.h></math.h></string.h></stdiib.h></stdio.h>	k & * \ * * * * * * * * * * * * * * * * *	
#include "wet9head.h"		
. void allocate_nsim(void)		
a_nnodes_x = (int*) malloc(nsim*sizeof(int)); a_nnodes_all = (int*) malloc(nsim*sizeof(int)); a_nnodes_all = (int*) malloc(nsim*sizeof(int)); a_key_int = (int*) malloc(nsim*sizeof(int)); a_key_int = (int*) malloc(nsim*sizeof(int)); a_key_interface = (int*) malloc(nsim*sizeof(int)); a_ney_scheme = (int*) malloc(nsim*sizeof(int)); a_niter_cycles = (int*) malloc(nsim*sizeof(int)); a_niter_cycles = (int*) malloc(nsim*sizeof(int)); a_niter_cycle = (int*) malloc(nsim*sizeof(int)); a_niter_cycle = (int*) malloc(nsim*sizeof(double)); a_length_x = (double*) malloc(nsim*sizeof(double)); a_length_x = (double*) malloc(nsim*sizeof(double)); a_delta_x = (double*) malloc(nsim*sizeof(double)); a_delta_x = (double*) malloc(nsim*sizeof(double)); a_force_x = (double*) malloc(nsim*sizeof(double)); a_force_y = (double*) malloc(nsim*sizeof(double)); a_cspeed2 = (double*) malloc(nsim*sizeof(double)); a_cspeed2 = (double*) malloc(nsim*sizeof(double)); a_nsszl = (double*) malloc(nsim*sizeof(double)); a_nszcol = (double*) malloc(nsim*sizeof(double)); a_nszcolleft = (double*) malloc(nsim*sizeof(double)); a_nszcolleft = (double*) malloc(nsim*sizeof(double)); a_uwall_bot = (double*) malloc(nsim*sizeof(double)); a_uwall_bot = (double*) malloc(nsim*sizeof(double)); a_uwall_left = (double*) malloc(nsim*sizeof(double)); a		
void free_nsim(void)		
<pre>free(a_nnodes_x); free(a_nnodes_y); free(a_nnodes_all); free(a_lambda); free(a_key_init); free(a_key_boundary); free(a_key_boundary); free(a_key_lnterface); free(a_key_lnterface); free(a_nnter_lnt); free(a_nnter_lnt);</pre>		

Jun 26 1999 20:56 wet9inout.c	out.c Page 2
free (a_length_X); free (a_length_Y); free (a_delta_X); free (a_delta_X); free (a_delta_X); free (a_force_X); free (a_force_Y); free (a_mass1); free (a_mass1); free (a_mass1); free (a_mass1); free (a_mass1); free (a_nasr0); free (a_uxwall_top); free (a_uxwall_bot); free (a_uxwall_left); free (a_uxwall_right); free (a_uxwall_right); free (a_uxwall_right); free (a_uxwall_right); free (a_uxwall_right);	·
void wet9_input (void)	
int 1, j; char dummy[128];	
/ double kboltz=1.381e-23, amu=1.661e-27, tem kboltz = J/(molecules*K), amu = (kg), temp * FIER *fin. *fros*	temp=300.00; nmp - (K);
<pre>iz, "program <wet9> out_name); ;;</wet9></pre>	stopped - input file <%s> does not exist !\n",
<pre>fscanf(fin, "%s %d\n", dummy, ensim); if(nsim < 1) exit(1);</pre>	
allocate_nsim();	
for (i=0; i <nsim; i++)<="" td=""><td></td></nsim;>	
<pre>fscanf(fin, "%s %d %s %d\n", dummy, &nnodes_x, dummy, &nnodes_y); fprint(fitez, "nnodes x= %d nnodes_y= %d\n", nnodes_y); fscanf(fin, "%s %d\n", dummy, λ); fprint(ffrez, "lambda= %d\n", lambda); fscanf(fin, "%s %lf %s %lf\n", dummy, δ_x, dummy, δ_y); if(delta_y) != delta_x; fprint(ffrez, "delta_x= %lf delta_y= %lf\n", delta_t= %lf\n", delta_t= %lf\n", delta_t= %20.151f\n", delta_t= %20.151f\n", delta_t= %20.151f\n", delta_t= %lf\n", key_boundary); fprint(ffrez, "delta_t= %20.151f\n", delta_t= %lf\n", key_boundary); fprint(ffrez, "key_init_ = %d % %d\n", dummy, &key_init_ = %d\n", key_boundary); fprint(ffrez, "key_init_ = %d\n", dummy, &key_scheme, dummy, &key_boundary); fprint(ffrez, "key_init_ = %d\n", dummy, &key_scheme, dummy, &key_init_ key_boundary); fscanf(fin, "%s %d %s %d\n", dummy, &key_scheme, dummy, &key_init_ key_boundary);</pre>	<pre>x, dummy, &nnodes_y); \n",nnodes_x, nnodes_y); a_x, dummy, δ_y); \n", delta_x, delta_y); \t.; \t.; \t.; \t.; \t.; \t.; \t.; \t.</pre>

wet9inout.c

wet9inout.c

Jun 26 1999 20:56

Jun 26 1999 20:56	wet9inout.c	Page 3
<pre>printf("key_interface=kd\n",key_interface); fprintf(frez, "key_scheme=kd\n",key_scheme); fscanf(fin, "%s & fd % & kd\n", dummy, fnrcyles du ka kd\n", fprintf(frez, "nrcyles= %d niter_cycle= %d noryles,niter_cycle,niter_cycle= %d noryles,niter_cycle,niter_cycle= %d noryles,niter_cycle,niter_inthinthinthinthinthinthinthinthinthinth</pre>	<pre>printf("key_interface=%d\n",key_interface); fprintf(frez,"key_scheme=%d\n",key_scheme); fscanf(fin."%s %d %s %d %s %d\n", fscanf(fin."%s %d %s %d %s %d %s %d %s %d %s %d %d</pre>	
ISCANT (III), "88 %11 %5 %11 fSCANT (fin, "%8 %1f %8 %1f' fSCANT (fin, "%8 %1f %8 %1f' nSEVOI = nzerolleft;	<pre>Cast (Lin, "%s %lf %s %lf(n" dummy, &mass, quummy, ecall;; fscanf(fin, "%s %lf %s %lf(n" dummy, &mass, dummy, &tau2); fscanf(fin, "%s %lf %s %lf(n", dummy, &nzerolleft, dummy, &nzero2left); fscanf(fin, "%s %lf %s %lf(n", dummy, &nzero1left, dummy, &nzero2left); nserol = nzero2left; nserol = nzero2left;</pre>	
f frez, mass]=%lf f(frez, mass]=%lf f(frez, mass]=%lf f(frez, nzerolleft= nzerolleft, nzero		
<pre>fprintf(frez, "naerolright=%lf fscanf (fin, "%s %lf %s %lf\n" fprintf(frez, "uxwall_bot=%lf fscanf(fin, "%s %lf %s %lf\n" fprintf(frez, "uxwall_top=%lf fscanf(fin, "%s %lf %s %lf\n" fprintf(frez, "uxwall_top=%lf fscanf(fin, "%s %lf %s %lf\n" fprintf(frez, "uxwall_left=%lf fprintf(frez, "uxwall_left=%lf fprintf(frez, "uxwall_left=%lf fprintf(frez, "uxwall_left=%lf)", fprintf(frez, "uxwall_left=%lf)</pre>	<pre>fprintf(frez, "nzerolright=%!f nzeroZright=%!f\n", fscanf(frez, "nzerolright=%!f\n", dummy, duxwall_bot, dummy, &uywall_bot); fscanf(fin, "% % %!f % % %!f\n", dummy, duxwall_bot, dummy, euywall_bot); fprintf(frez, "uxwall_bot=%!f uywall_bot=%!f\n", uxwall_bot; fprintf(frez, "uxwall_top=%!f uywall_top=%!f\n", uxwall_top, uywall_top); fscanf(fin, "% % %!f % % %!f\n", dummy, &uxwall_lop; fscanf(fin, "% % %!f % % %!f\n", dummy, &uxwall_left, uywall_left); fprintf(frez, "uxwall_left=%!f uywall_left=%!f\n", dummy, &uywall_left); fprintf(frez, "uxwall_left=%!f uywall_left);</pre>	
fscanf(fin,*s, %if %s %if\" fprintf(frez,"uxaal,_right=%if fscanf(fin,*s, %if %s %if\" dummy, %force); force, x = 1.0e-20; force, x = 1.0e-20; gforce *= 1.0e-20; fprintf(frez,"force,x=%e force fprintf(frez,"force,x=%e)		
<pre>if(mass1) cspeed1 = (3.00 * kboltz * t else cspeed1 = 1.0000; if(mass2) cspeed2 = (3.00 * kboltz * t else cspeed2 = 1.0000; cspeed1 = sqrt(cspeed1); cspeed2 = sqrt(cspeed1); fprintf(frez,*cspeed1=\$\$25.20e</pre>	<pre>z * temp) / (mass1 * amu); z * temp) / (mass2 * amu); v.20e cspeed2=%25.20e\n", cspeed1, cspeed2);</pre>	
<pre>/* if(key_scheme == 6) delta_x = delta_y = cspeed1 if(cspeed2*tau2 > delta_x)</pre>	* tau1; 2 * tau2;	
<pre>fprintf(frez."key_scheme=%d delta_x=</pre>	<pre>scheme=%d delta_x= %lf delta_y= %lf\n", ,delta_x,delta_y); 1 < delta_y </pre>	

*/
<pre>vet9> stopped - delta_t > tau 1 or tau2 !\n"); rez,"<wet9> stopped - delta_t > tau 1 or tau2 !\n"); 32);</wet9></pre>
/ / // if(cspeed1 > cspeed2) delta_t = delta_x / cspeed1; else delta t = delta_x / cspeed2;
<pre>l = nnodes_x = delta_x * (= delta_y * (</pre>
<pre>fprintf(frez,"%d %d %lf %lf %e\n",nnodes_x,nnodes_y,</pre>
<pre>a_nnodes_x[1] = nnodes_x; a_nnodes_x[1] = nnodes_y; a_nnodes_all[1] = nnodes_all; a_lambda[1] = lambda; a_key_init[1] = key_init[t]; a_key_init[1] = key_houndary;</pre>
<pre>a_key_scheme[i] = key_scheme; a_key_interface; a_ncyles[i] = ncyles; a_nter_cycle[i] = niter_cycle; a_niter_init[i] = niter_init;</pre>
a_length_x! = length_x; a_length_y! = length_y; a_delta_x[i] = delta_x; a_delta_y[i] = delta_y; a_delta_t[i] = delta_t;
a_force_x[i] = force_x; a_force_y[i] = force_y; a_gforce[i] = gforce; a_massl[i] = massl;
a_mass[l] = mass[, a_cspeed[] = cspeed; a_cspeed2[] = cspeed2; a_tau1[1] = tau1; a_tau1[] = tau1;
cauz,
k
<pre>a_uxvall_right[i] = uxwall_right; a_uywall_right[i] = uywall_right; fclose(fin); fclose(fin);</pre>

9

Page 5	20_\$9.9d_\$9
Jun 26 1999 20:56 wet9inout.c	
Jun	Void Void Linea and Line

nec.utt.ro	
umag3.m	
a from f	
sofone	
Printed by	

Jun 26 1999 13:52	Page 1
/*************************************	
<pre>#include <stdlib.h> #include <math.h> #include <stdlo.h></stdlo.h></math.h></stdlib.h></pre>	
<pre>#include "wet9head.h" void init_lattice_functions(void)</pre>	
file (double*) malloc (modes_all*sizeof(double)); fil = (double*) mal	
double*) malloc(innodes_all*sizeof = (double*) malloc(innodes_all*sizeof	

Jun 26 1999 13:52 Wet9aux.c	Page 2
sourcell = (double*) malloc (nnodes_all*sizeof (double)) sourcels = (double*) malloc (nnodes_all*sizeof (double)) sill = (d	(e)); (e
<pre>void free_lattice_functions(void) free(f10); free(f11); free(f12); free(f13); free(f14);</pre>	

wet9aux.c

O	
Ξ.	
≝	
⇉	
ပ္ရ	
உ	
=	
က်	
Ō	
<u> </u>	
☱	
flumag3.mec.utt.ro	
Ξ	
Ĕ	
5	
w	
an .	
9	
one	
otone	
sofone	
y sofonea	
ි	
ි	
ි	
ි	
ි	
ි	
ठ	

wet9aux.c

free (f16);			10000
free(f16);			
			free(sfl3);
free(11);			free (sf15);
free(f20);			free(sf16);
free(f21);			free(stl/);
free(f23);			free(sf20);
free(f24);			free(sf21);
iree(123);			free(sf23);
free(f27);			free(sf24);
free(f28);			free(sf25);
free(ff10);			free(S12b);
free(IIII);			free(sf28);
free (ff13);			free (uxloc);
free(ff14);			free (uyloc);
free(ff15);			free (uxloc1);
free(ff16);			free (uvicei);
free(III/);			free (ux10c2);
free(ff20);			free (nv1);
free(ff21);			free (nv2);
free(ff22);			free (nv3);
free(ff23);			free (nv4);
free (ff24);			free (nv5);
free(ff25);			free (nvo);
rree(rrzb);		_	froo (nv.)
Iree(II2/);			free (boundary mode)
free (neg10):			free(nloc1);
free (neg11):			free(nloc2);
free (neq12);			free (colorfield);
free (neg13);			free (gradn1x);
free (neg14);			free (gradn2x);
free (neq15);			free (graduly);
free (neq16);			ree (gradicy);
free (ned 1) ;		, .	
free (neq20);			
free (neg21);			void test_distribution
free (neq22);			_
free (neq23);			
Iree(neq24);			FILE *frez:
free (neq26);			int k;
free (neq27);			frez = fopen(rez_
free (neq28);			for (k=0; k <nnodes_a< td=""></nnodes_a<>
free(source10);			1 4 (- 60 (1)-1 / 0 /
free (sourcell);			0 / [4]0411 74
free (source12);			forintf(fre
free (source14):			146
free (source15);			exit(1);
free (source16);			
free (source17);			1f(nf1[k] < 0.0
free (source18);			4 4 5 1 5 5 5 5
free(source20);			The increase of the
free (source22);	•		exit(1);
free (source23);		-	
free (source24);			if(nf2[k] < 0.0
free(source25);			forintf(fr
free (source27);			it
free (source28);			exit(1);
free(sfl0);		-	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
free(sill);			,

free (8f21); free (sf22);	
free(sf23); free(sf24);	
<pre>free(sf25); free(sf26);</pre>	
free(sf27); free(sf28);	
free (urloc); frac(urloc);	
<pre>free(ulloc); free(uxloc1);</pre>	
<pre>free(uyloc1); free(iyloc2);</pre>	
uyloc2)	
<pre>free(nv1); free(nv2):</pre>	
free(nv3);	
<pre>free(nv4); free(nv5);</pre>	
free(nv6);	
free(nv); free(nv);	
free (boundary mode);	
free(nloc1);	
free(colorfield);	
free (gradnix);	
free (gradnly);	
<pre>free(gradn2y); }</pre>	
<pre>void test_distribution_functions(double nf0[], double nf1[], double nf2[],</pre>	
FILE *frez;	
<pre>rez = topen(rez_name, aw'); for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>	
if(nf0[k] < 0.00)	
<pre>fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf0=%g\n",</pre>	
<pre>iter,k,nf0(k)); exit(1);</pre>	
) 1f(nf1[k] < 0.00)	
O CONTRACTOR (TO)	
<pre>iprint:(rrez, \n\nNsGAIIVE F: iter=%d K=%d nil=%g\n", iter,k,nfl[k]);</pre>	
exit(1);	
if(nf2[k] < 0.00)	
<pre>fprintf(frez,"\n\nNEGATIVE F: iter=%d k=%d nf2=%g\n", iter,k,nf2[k]);</pre>	
exit(1);	
if[nf3[k] < 0.00)	
_	

wet9init.c

Jun 25 1999 13:31

Page 2

	一般安全在安全的安全的大学的大学的
* wet9init.c arrays initialisation	/ 由于有关的基础的对象的对象的对象的。 * * * * * * * * * * * * * * * * * * *
<pre>#include <stdio.h> #include <stdib.h> #include <math.h> #include <math.h> #include <string.h></string.h></math.h></math.h></stdib.h></stdio.h></pre>	
<pre>void print_tavec(int Kk, int nv[]) { int i, j;</pre>	
{	
<pre>fprintf(fout, "%3d ", nv[(j-1)*nnodes_x+i]); }</pre>	
fclose(fout);	
, void getavec_square(void)	
<pre>int i, j, k; k = 0; for(j=0; j<nnodes_y; j++)<="" pre=""></nnodes_y;></pre>	
{ for(i=0; i <nnodes_x; i++)<="" td=""><td></td></nnodes_x;>	
f(i == (nnodes_x-1)	*d *d\n", nv6[k],nv7[k],nv8[k]);

 void init_ecx_nine(void)	
 int i;	
<pre>double xminusminus, xminus, xzero, xplus, xx, xy; double lxminusminus, lxminus, lxzero, lxplus,</pre>	
ww(0) = w0; ww(1) = w1; ww(3) = w1; ww(4) = w1; ww(5) = w2; ww(7) = w2; ww(8) = w2;	•
ecx[0] = ((double) 0); ecy[0] = ((double) 0); ecy[1] = ((double) 1); ecy[1] = ((double) 0); ecy[2] = ((double) 0); ecy[2] = ((double) 1); ecx[3] = ((double) -1);	
 (double) 1) (double) 1]	
(i=0; i < 9; i++)	-
<pre>{ ecx[i] * cspeedl; ecy1[i] = ecy[i] * cspeedl; ecx2[i] = ecx[i] * cspeed2; ecy2[i] = ecy[i] * cspeed2;</pre>	
<pre>ecxforcel[i] = delta_t * force_x * ecx[i] * cspeed1 / (kboltz * temp); ecyforcel[i] = delta_t * force_y * ecy[i] * cspeed1 / (kboltz * temp); ecyford1[i] = ecxforcel[i] + ecyforcel[i]; ecxforce2[i] = delta_t * force_x * ecx[i]; ecyforce2[i] = delta_t * force_y * ecy[i] * cspeed2 / (kboltz * temp); ecyforce2[i] = delta_t * force_y * ecy[i] * cspeed2 / (kboltz * temp); }</pre>	
 xminus = -1.000; xzero = 0.000; xplus = 1.000;	
for(1=1; 1<9; 1++)	
<pre>'xx = xzero - eox1[i] * delta_t / delta_x; xy = xzero - eoy1[i] * delta_t / delta_x;</pre>	
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	

9

Jun 25 1999 13:31 wet9init.c	Page 3	
<pre>lyminus = ((xy-xzero)*(xy-xplus))/((xminus-xzero)*(xminus-xplus)); lyzero = ((xy-xminus)*(xy-xplus))/((xzero-xminus)*(xzero-xplus)); lyplus = ((xy-xminus)*(xy-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		
nc10[i] = lxzero * lyzero; nc11[i] = lxplus * lyzero; nc12[i] = lxzero * lyplus; nc13[i] = lxminus * lyzero; nc14[i] = lxzero * lyminus; nc15[i] = lxplus * lyplus; nc15[i] = lxminus * lyplus; nc17[i] = lxminus * lyminus; nc17[i] = lxminus * lyminus; nc18[i] = lxminus * lyminus;		
xx = xzero - ecx2[i] * delta_t / delta_x; xy = xzero - ecy2[i] * delta_t / delta_x;		
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		
<pre>lyminus = ((xy-xzero)*(xy-xplus))/((xminus-xzero)*(xminus-xplus)); lyzero = ((xy-xminus)*(xy-xplus))/((xzero-xminus)*(xzero-xplus)); lyplus = ((xy-xminus)*(xy-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		
nc20[1] = lxzero * lyzero; nc21[1] = lxplus * lyzero; nc22[1] = lxzero * lyplus; nc22[1] = lxzero * lyplus; nc24[1] = lxminus * lyzero; nc24[1] = lxzero * lyminus; nc25[1] = lxplus * lyplus; nc25[1] = lxminus * lyplus; nc26[1] = lxminus * lyminus; nc28[1] = lxplus * lyminus;		
xminus = -1.000; xzero = 0.000; xplus = 1.000; yminus = -1.000; yzero = 0.000; yplus = 1.000;		
for(i=1; i<9; i++)		
<pre>'xx = xzero - ecx1[i] * delta_t / delta_x; xy = xzero - ecy1[i] * delta_t / delta_x;</pre>		
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	,	
<pre>lyzero = (xy-yplus)/(yzero-yplus); lyplus = (xy-yzero)/(yplus-yzero);</pre>		
<pre>ncbot10[1] = lxzero * lyzero; ncbot11[1] = lxplus * lyzero; ncbot12[1] = lxzero * lyplus; ncbot13[1] = lxzero * lyplus; ncbot13[1] = lxplus * lyplus; ncbot16[1] = lxplus * lyplus;</pre>		
<pre>xx = xzero - ecx2[i] * delta_t / delta_x; xy = xzero - ecy2[i] * delta_t / delta_x;</pre>		
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxeero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		

Jun 25 1999 13:31 wet9init.c	Page 4
<pre>lyzero = (xy-yplus)/(yzero-yplus); lyplus = (xy-yzero)/(yplus-yzero);</pre>	
<pre>ncbot20[1] = lxzero * lyzero; ncbot21[1] = lxplus * lyzero; ncbot22[1] = lxzero * lyplus; ncbot23[1] = lxminus * lyzero; ncbot25[1] = lxmlus * lyplus; ncbot26[1] = lxminus * lyplus;</pre>	
<pre>xx = xzero - ecx1[i] * delta_t / delta_x; xy = xzero - ecy1[i] * delta_t / delta_x;</pre>	
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	
<pre>lyzero = (xy-yminus) / (yzero-yminus); lyminus = (xy-yzero) / (yminus-yzero);</pre>	
<pre>nctopl0[i] = lxzero * lyzero; nctopl1[i] = lxplus * lyzero; nctopl3[i] = lxminus * lyzero; nctopl7[i] = lxminus * lyminus; nctopl4[i] = lxminus * lyminus; nctopl4[i] = lxxero * lyminus;</pre>	
<pre>xx = xzero - ecx2[i] * delta_t / delta_x; xy = xzero - ecy2[i] * delta_t / delta_x;</pre>	
<pre>lminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	
<pre>lyzero = (xy-yminus) / (yzero-yminus); lyminus = (xy-yzero) / (yminus-yzero);</pre>	
<pre>nctop20[i] = lxzero * lyzero; nctop21[i] = lxplus * lyzero; nctop23[i] = lxminus * lyzero; nctop27[i] = lxminus * lyminus; nctop24[i] = lxzero * lyminus; nctop24[i] = lxzero * lyminus;</pre>	
<pre>xx = xzero - ecx1[i] * delta_t / delta_x; xy = xzero - ecy1[i] * delta_t / delta_x;</pre>	
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	
<pre>lxzero = (xx-xplus)/(xzero-xplus); lxplus = (xx-xzero)/(xplus-xzero);</pre>	
<pre>ncleft15[i] = lxplus * lyplus; ncleft11[i] = lxplus * lyzero; ncleft18[i] = lxplus * lyzero; ncleft2[i] = lxzero * lyplus; ncleft10[i] = lxxero * lyplus; ncleft10[i] = lxxero * lyzero; ncleft14[i] = lxxero * lyminus;</pre>	
<pre>xx = xzero - ecx2[i] * delta_t / delta_x; xy = xzero - ecy2[i] * delta_t / delta_x;</pre>	
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)) lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	
lxzero = (xx-xplus)/(xzero-xplus);	

=

_	
I#.ro	
9C.L	
3.me	
nagg	
<u>E</u>	
Ē	
fro	
nea	
ofon	
by s	
g	
inte	
ď.	

Jun 25 1999 13:31 wet9init.c	Page 5	Jun 25 1999 13:31
lxplus = (xx-xzero)/(xplus-xzero);		nul4[i] = lxzero
lxplus * lxplus * lxplus * lxplus *		
M LXZero M LXZero M LXZero		i=2; xx = xzero - ecx1[xv = vr]us - ecv1[
<pre>xx = xzero - ecx1[i] * delta_t / delta_x; xy = xzero - ecy1[i] * delta_t / delta_x;</pre>		23
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>		ı " " " " " " " " " " " " " " " " " " "
<pre>lxzero = (xx-xminus) / (xzero-xminus); lxminus = (xx-xzero) / (xminus-xzero);</pre>		
<pre>ncright10[1] = lxzero * lyzero; ncright12[1] = lxzero * lyplus; ncright14[1] = lxzero * lyminus; ncright16[1] = lxminus * lyplus; ncright13[1] = lxminus * lyzero; ncright17[1] = lxminus * lyminus;</pre>	. 1 89	nul(= xxero nul(= xxero nul(= xxero nul(= xxero nul(= xxero nul(= xxero nul(= xxero nul(= xxero
<pre>xx = xzero - ecx2[i] * delta_t / delta_x; xy = xzero - ecy2[i] * delta_t / delta_x;</pre>		
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>		inu
<pre>lxcero = (xx-xminus) / (xxero-xminus); lxminus = (xx-xzero) / (xminus-xzero);</pre>		lxminus = (xx-xze lxzero = (xx-xmir lxplus = ((xx-xmir
lxzero lxzero lxzero lxminus		11 11
11		H H H
xminus = -1.000; xzero = 0.000; xplus = 1.000;		nul4[i] = lxzero nul5[i] = lxplus nul6[i] = lxminus nul7[i] = lxminus nul7[i] = lxminus
yminus = -1.000; yzero = 0.000; yplus = 1.000;		sero
<pre>i=1; xx = xplus - ecxl[i] * delta_t / delta_x; xy = yzero - ecyl[i] * delta_t / delta_x;</pre>		
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>		
nul0[i] = lxzero * lyzero; nul1[i] = lxplus * lyzero; nul2[i] = lxzero * lyplus; nul3[i] = lxminus * lyzero;		

<pre>[1] = lxzero * lyminus; [1] = lxplus * lyplus; [1] = lxminus * lyplus; [1] = lxminus * lyminus; [1] = lxminus * lyminus; [1] = lxplus * lyminus;</pre>	<pre>xzero - ecx1[i] * delta_t / delta_x; yplus - ecy1[i] * delta_t / delta_x;</pre>	<pre>nus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); rro = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xplus-xplus)); us = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	<pre>nus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); ro = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); us = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	[i] = lxzero * lyzero; [ii] = lxplus * lyzero; [ii] = lxzero * lyplus; [ii] = lxzero * lyminus; [ii] = lxplus * lyplus; [ii] = lxplus * lyplus; [ii] = lxminus * lyplus; [ii] = lxminus * lyminus; [ii] = lxminus * lyminus; [ii] = lxplus * lyminus;); = xminus - ecx1[i] * delta_t / delta_x; = yzero - ecy1[i] * delta_t / delta_x;	<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	[[1] = lxzero * lyzero; [[1] = lxplus * lyzero; [[1] = lxzero * lyplus; [[1] = lxminus * lyzero; [[1] = lxplus * lyplus; [[1] = lxplus * lyplus; [[1] = lxminus * lyminus; [[1] = lxplus * lyminus;	<pre>* xzero - ecx1[i] * delta_t / delta_x; = yminus - ecy1[i] * delta_t / delta_x;</pre>	<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	
		lxminus = (lxzero = ((lxplus = ((lyminus = (lyzero = ((lyplus = ((1=3; xx = xminus xy = yzero	lxminus = (lxzero = ((lxplus = ()	lyminus = (lyzero = ((lyplus = ()		i=4; xx = xzero - xy = yminus	lxminus = () lxzero = () lxplus = ()	lyminus = (lyzero = (lyplus = (

Ę
c.utt
41
.mec
lag3
2
fluma
Ē
Į.
=
<u>@</u>
Ö
sofone
>
<u>0</u>
ĕ
Ë
ď

wet9init.c

Jun 25 1999 13:31 wet9init.c	Page 7	Jun 25 1999 13:31
<pre>nul7(i) = lxminus * lyminus; nul8[i] = lxplus * lyminus;</pre>		i=8; xx = xplus - ecx1[xv = vminus - ecv]
<pre>i=5; xx = xplus - ecx1[i] * delta_t / delta_x; xy = yplus - ecy1[i] * delta_t / delta_x;</pre>	•	x)) = 00
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		lyminus = ((xx-xmin lyminus = ((xy-yze lygero = ((xy-ymin lygero = ((xy-ymin
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>		. " " '
null[i] = lxzero * lyzero; null[i] = lxplus * lyzero; null[i] = lxero * lyplus; null[i] = lxero * lyplus; nulf[i] = lxero * lyminus; nulf[i] = lxero * lyminus; nulf[i] = lxminus * lyplus; nulf[i] = lxminus * lyplus; nulf[i] = lxminus * lyminus; nulf[i] = lxminus * lyminus;		til = 1xx til = 1xx til = 1xx til = 1xx til = 1xx til = 1xx xplus -
<pre>i=6; xx = xminus - ecx[[1] * delta_t / delta_x; xy = yplus - ecyl[1] * delta_t / delta_x;</pre>		<pre>xy = yzero - ecy2 lxminus = ((xx-xze lxzero = ((xx-xminus))</pre>
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		Lxplus = ((xx-xml) Lyminus = ((xy-yze Lyminus) Lyminus
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>		
nul0[1] = lx2ero * lyzero; nul1[1] = lxplus * lyzero; nul2[1] = lxzero * lyplus; nul3[1] = lxzero * lyplus; nul4[1] = lxzero * lyplus; nul5[1] = lxzero * lyplus; nul6[1] = lxminus * lyplus; nul6[1] = lxminus * lyplus; nul7[1] = lxminus * lyplus; nul7[1] = lxminus * lyminus; nul8[1] = lxminus * lyminus;		
<pre>i=7; xx = xminus - ecx1[i] * delta_t / delta_x; xy = yminus - ecy1[i] * delta_t / delta_x;</pre>		n To
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>		
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>		H H I
null[i] = lxzero * lyzero; null[i] = lxplus * lyzero; null[i] = lxzero * lyplus; null[i] = lxzero * lyplus; null[i] = lxzero * lyminus; null[i] = lxzero * lyminus; null[i] = lxplus * lyplus; null[i] = lxplus * lyplus;		
= lxminus *		i=3; xx = xminus - ecx

<pre>s - ecx1[1] * delta_t / delta_x; us - ecy1[i] * delta_t / delta_x;</pre>	<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero)); lxminus = ((xy-yrero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xx-yplus))/((yzero-yminus)*(xzero-yplus));</pre>	((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero)); Lxzero * lyzero; Lxzero * lyplus;	<pre>i=1; xx = xplus - ecx2[i] * delta_t / delta_x; xy = yzero - ecy2[i] * delta_t / delta_x; xy = yzero - ecy2[i] * (kita_t / delta_x;</pre>	<pre>((xy-yzezo)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	Ixzero * lyzero; Ixplus * lyzero; Ixzero lyplus; Ixminus * lyplus; Ixxero * lyminus; Ixxero * lyplus; Ixminus * lyplus; Ixminus * lyplus; Ixminus * lyminus; Ixplus * lyminus;	<pre>co - ecx2[i] * delta_t / delta_x; ls - ecy2[i] * delta_t / delta_x;</pre>	<pre>((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	<pre>((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	Ixzero * lyzero; Ixzero * lyzero; Ixzero * lyzero; Ixzero * lyplus; Ixminus * lyplus; Ixminus * lyplus; Ixminus * lyplus; Ixminus * lyminus; Ixminus * lyminus *	
Ψ,	lxminus = ((xx-xze lxzero = ((xx-xmin lxplus = ((xx-xmin lyminus = ((xy-yze lyzero = ((xy-yze	lyplus = ((xy-ymin nul0[1] = lxZero nul1[1] = lxZero nul2[1] = lxZero nul3[1] = lxZero nul4[1] = lxZero nul5[1] = lxZero nul7[1] = lxZero nul7[1] = lxZero nul8[1] = lxZero nu	i=1; xx = xplus - ecx2[xy = yzero - ecy2[lxminus = ((xx-xze lxzero = ((xx-xmin lxplus = ((xx-xmin lxplus = (xx-xmin	lyminus = ((xy-yze lyzero = ((xy-ymin lyplus = ((xy-ymin	nu20[i] = lxzero nu21[i] = lxplus nu22[i] = lxzero nu23[i] = lxminus nu24[i] = lxzero nu25[i] = lxplus nu26[i] = lxminus nu26[i] = lxminus nu28[i] = lxminus	i=2; xx = xzero - ecx2 xy = yplus - ecy2	lxminus = ((xx-xze lxzero = ((xx-xmir lxplus = ((xx-xmir	lyminus = ((xy-yze lyzero = ((xy-ymin lyplus = ((xy-ymin	nu20[1] = lxzero nu21[1] = lxplus nu22[1] = lxeero nu23[1] = lxminus nu24[1] = lxzero nu25[1] = lxplus nu27[1] = lxplus nu27[1] = lxminus	i=3; xx = xminus - ecx;

Jun 25 1999 13:31 wet9init.c	Page 10
<pre>lxplus = ((xx-xminus) *(xx-xzero))/((xplus-xminus) *(xplus-xzero));</pre>	
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	
nu20[i] = lxzero * lyzero; nu21[i] = lxplus * lyzero; nu22[i] = lxero * lyplus; nu23[i] = lxminus * lyzero; nu24[i] = lxero * lyminus; nu25[i] = lxplus * lyplus; nu25[i] = lxminus * lyplus; nu27[i] = lxminus * lyminus; nu28[i] = lxplus * lyminus;	
<pre>i=7; xx = xminus - ecx2[i] * delta_t / delta_x; xy = yminus - ecy2[i] * delta_t / delta_x;</pre>	
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	
nu20[1] = lxzero * lyzero; nu2[1] = lyplus * lyzero; nu2[1] = lxzero * lyplus; nu2[1] = lxzero * lyplus; nu2[4] = lxzero * lyminus; nu2[5] = lxplus * lyplus; nu2[6] = lxminus * lyplus; nu2[1] = lxplus * lyminus; nu2[1] = lxplus * lyminus;	
i=8; xx = xplus - ecx2[i] * delta_t / delta_x; xy = yminus - ecy2[i] * delta_t / delta_x;	
<pre>lxminus = ((xx-xzero)*(xx-xplus))/((xminus-xzero)*(xminus-xplus)); lxzero = ((xx-xminus)*(xx-xplus))/((xzero-xminus)*(xzero-xplus)); lxplus = ((xx-xminus)*(xx-xzero))/((xplus-xminus)*(xplus-xzero));</pre>	
<pre>lyminus = ((xy-yzero)*(xy-yplus))/((yminus-yzero)*(yminus-yplus)); lyzero = ((xy-yminus)*(xy-yplus))/((yzero-yminus)*(yzero-yplus)); lyplus = ((xy-yminus)*(xy-yzero))/((yplus-yminus)*(yplus-yzero));</pre>	
nu20[1] = lxzero * lyzero; nu21[1] = lxplus * lyzero; nu22[1] = lxzero * lyplus; nu23[1] = lxminus * lyzero; nu24[1] = lxzero * lyminus; nu25[1] = lxplus * lyplus; nu25[1] = lxplus * lyplus; nu25[1] = lxplus * lyplus; nu27[1] = lxminus * lyplus; nu27[1] = lxminus * lyminus;	
/* xminusminus = -1.000; xminus = 0.000; xzero = 1.000;	
for(i=1, 1<9; i++)	
<pre>'xx = xzero - ecx1[i] * delta_t / delta_x; xy = xzero - ecy1[i] * delta_t / delta_x;</pre>	

Jun 25 1999 13:31	wet9init.c	Page 11
<pre>lxminusminus = ((xx-xx lxminus = ((xx-xminus) lxminus = ((xx-xminus) lxzero = ((xx-xminus) lxzero = ((xx-xminus)</pre>	<pre>lnus = ((xx-xminus)*(xx-xzero))/ ((xminusminus-xminus)*(xx-xzero)); = ((xx-xminusminus)*(xx-xzero))/ ((xminus-xminusminus)*(xminus-xzero)); ((xx-xminusminus)*(xminus))/ ((xzero-xminusminus)*(xzero-xminus));</pre>	
<pre>lyminusminus = ((xy-xx (xxminus) lyminus = ((xy-xminus) ((xminus-xmi) lyzero = ((xy-xminusm) ((xzero-xminusm))</pre>	<pre>Lnus = ((xy-xminus)*(xy-xzero))/ ((xminusminus-xminus)*(xminusminus-xzero)); = ((xy-xminusminus)*(xy-xzero))/ ((xxinus-xminusminus)*(xy-xminus))/ ((xy-xminusminus)*(xy-xminus))/ ((xzero-xminusminus)*(xzero-xminus));</pre>	
null[i] = lxminus * lym null[i] = lxzero * lymi null[i] = lxminus * lym null[i] = lxminus * lym null[i] = lxminus * lym null[i] = lxzero * lyze null[i] = lxminus * lym null[i] = lxzero * lyze null[i] = lxzero * lyminus	lxminus * lyminus; lxzero * lyminus; lxxero * lyminus; lxminus * lyeero; lxminus * lyminus; lxminus * lyzero; lxminusminus * lyminusminus; lxzero * lyminusminus;	
xx = xzero - ecx2[i] xy = xzero - ecy2[i]	<pre>* delta_t / delta_x; * delta_t / delta_x;</pre>	
<pre>lxminusminus = ((xminu lxminus = ((xx-xminus-xmi (xminus-xmi lxzero = ((xx-xminusmi lxzero-xminusm)</pre>	<pre>lnus = ((xx-xminus)*(xx-xzero))/ ((xminusminus-xminus)*(xminusminus-xzero)); = ((xx-xminusminus)*(xx-xzero))/ ((xminus-xminusminus)*(xx-xminus); ((xx-xminusminus)*(xx-xminus))/ ((xzero-xminusminus)*(xzero-xminus));</pre>	
<pre>lyminusminus = ((xy-x (xminus lyminus = ((xy-xminus-xmi ((xminus-xmi lyzero = ((xy-xminusmi lyzero = ((xy-xminusmi lyzero = ((xy-xminusmi lyzero = ((xy-xminusmi lyzero = (xy-xminusmi lyzero = (</pre>	<pre>funs = ((xy-xminus) *(xy-xzero))/ ((xminusminus -xminus) *(xminusminus -xzero)); ((xminus-xminus) *(xy-xzero))/ ((xminus-xminusminus) *(xminus-xzero)); ((xy-xminusminus) *(xy-xminus))/ ((xzero-xminusminus) *(xzero-xminus));</pre>	
nu20[1] = lxminus * lyminus; nu21[1] = lxzero * lyminus; nu22[1] = lxminus * lyezro; nu22[1] = lxminus * lyminus; nu24[1] = lxminus * lyminus; nu25[1] = lxzero * lyzero; nu26[1] = lxminusminus * lyminus; nu27[1] = lxminusminus * lyminusminus * lyminusminusminusminusminusminusminusminus	lxminus * lyminus; lxzero * lyminus; lxminus * lyminus; lxminus * lyminus; lxminus * lyminusminus * lyminusminus * lyminusminus * lyminusminus * lyminusminus * lyzero; lxminusminus * lyzero; lxminusminus * lyminusminus;	
/ / serendip */		
xtl = -1.000; xtr = 1.000; xbl = -1.000; xbr = 1.000;		
ytl = 1.000; ytr = 1.000; ybl = -1.000; ybr = -1.000;		
for(i=1; i<9; i++)		
{ switch(1) {		

Printed by sofonea from flumag3.mec.utt.ro

Jun 25 1999 13:31 wet9init.c	Page 12
<pre>case 1: case 5: xx = xtr - 2.000 * ecx1[1] * delta_t / delta_x; xy = ytr - 2.000 * ecy1[1] * delta_t / delta_x; break; case 2:</pre>	
0	
case 7: xx = xbl - 2.000 * ecx1[i] * delta_t / delta_x; xy = ybl - 2.000 * ecy1[i] * delta_t / delta_x; break; case 4:	
case 8:	
<pre>printf("il=%d xx=%e xy=%e\n",i,xx,xy); */</pre>	
nsiti[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; nsitr[i] = (1.000 + xx*xtl)*(1.000 + xx*ytl) / 4.000; nsibi[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000; nsibr[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;	
/* printf("tl=%e tr=%e bl=%e br=%e\n",	
switch(i)	
<pre>case 1: case 5:</pre>	
case 7:	
case 8: xx = xbr - 2.000 * ecx2[i] * delta_t / delta_x; xy = ybr - 2.000 * ecy2[i] * delta_t / delta_x; break;	
/* printf("12=%d xx=%e xy=%e\n",i,xx,xy);	
ns2tl[1] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; ns2tr[1] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; ns2tl[1] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; ns2tr[1] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000;	

Printed by sofonea from flumag3.mec.utt.ro

wet9init.c

5 1999 13:31
((xt-xtl)*(ytr-ybr); (xt-xtr)*(yt-ybl); ((xbr-xbl)*(ybr-ytr); ((xbl-xbr)*(ybl-ytl));
* delta_t / delta_x; * delta_t / delta_x;
* delta_t / delta_x; * delta_t / delta_x;
+ xy*ytl) / 4.000; + xy*ytr) / 4.000; + xy*ybl) / 4.000; + xy*ybr) / 4.000;
((xtr-xtl)*(ytr-ybr)); ((xtl-xtr)*(ytl-ybl)); ((xbr-xbl)*(ybr-ytr)); ((xbl-xbr)*(ybl-ytl));
/ delta_x; / delta_x;
delta_t / delta_x; delta_t / delta_x;
xy*ytl) / 4.000; xy*ytr) / 4.000; xy*ybl) / 4.000; xy*ybl) / 4.000;
((xtr-xtl)*(ytr-ybr)); ((xtl-xtr)*(ytl-ybl)); ((xbr-xbl)*(ybr-ytr)); ((xbl-xbr)*(ybl-ytr));

1110	
<pre>case 4: case 5: xx = xbl - 2.000 * ecx1[i] * delta_t / delta_x; xy = ybl - 2.000 * ecy1[i] * delta_t / delta_x; break; default: break; }</pre>	
nslltl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; nslltr[i] = (1.000 + xx*xtt)*(1.000 + xy*ytt) / 4.000; nsllbl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000; nsllbr[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;	
<pre>nllltr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr)); nlltt[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl)); nllbr[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr)); nllbl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytr));</pre>	
<pre>switch(i) { case 2: case 7: xx = xtr - 2.000 * ecx1[i] * delta_t / delta_x; xy = ytr - 2.000 * ecy1[i] * delta_t / delta_x; break; break;</pre>	
case 6: xx = xbr - 2.000 * ecx![i] * delta_t / delta_X; xy = ybr - 2.000 * ecy![i] * delta_t / delta_X; break; default: break;	
<pre>nsrltl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; nsrltr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000; nsrlbl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000; nsrlbr[i] = (1.000 + xx*xbr)*(1.000 + xy*ybl) / 4.000;</pre>	
<pre>nlritr[i] = (xx-xtl)*(xy-ybr) / (xtr-xtl)*(ytr-ybr)); nlribr[i] = (xx-xtr)*(xy-ybl) / (xtl-xrn)*(ytl-ybl)); nlribr[i] = (xx-xbr)*(xy-ytr) / ((xbr-xbl)*(ybr-ybr)); nlribl[i] = (xx-xbr)*(xy-ytr) / ((xbl-xbr)*(ybl-ytr));</pre>	
switch(i) { case 2:	
case 5: xx = xbl - 2.000 * ecx2[1] * delta_t / delta_x; xy = ybl - 2.000 * ecy2[1] * delta_t / delta_x; break; case 1:	
<pre>case 6:</pre>	
<pre>nsb2t1[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; nsb2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000; nsb2b1[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000; nsb2br[i] = (1.000 + xx*xbr)*(1.000 + xy*ybr) / 4.000;</pre>	

ı#.ro	
3.mec.utt.	
ag3.r	
jum'j	
from 1	
Ø	
ofone	
by s	
ted	
Prin	

wet9init.c

Jun 25 1999 13:31 wet9init.c	Page 15	Jun 25 1999 13:31
<pre>nlb2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr)); nlb2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl)); nlb2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr)); nlb2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));</pre>		** **
switch(1) { case 3:	,	nsr2tr[1] = nsr2tr[1] = nsr2b1[1] = nsr2bz[1] =
<pre>case 8: xx = xt1 - 2.000 * ecx2[i] * delta_t / delta_x; xy = yt1 - 2.000 * ecy2[i] * delta_t / delta_x; break; case 4:</pre>		nlr2tr[i] = nlr2t1[i] = nlr2br[i] = nlr2bl[i] =
case /: xx = xtr - 2.000 * ecx2[i] * delta_t / delta_x; xy = ytr - 2.000 * ecy2[i] * delta_t / delta_x; break;		
<pre>default: break; }</pre>		VOIG INICATEDS.TIN FILE *fsav, *frez; int i.i.k. key rho
nst2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; nst2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytl) / 4.000; nst2bl[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000; nst2br[i] = (1.000 + xx*xbl)*(1.000 + xy*ybl) / 4.000;		double rand_coef = double radius, xc, frez=fopen(rez_nam
<pre>nlt2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr)); nlt2tl[i] = (xx-xtr)*(xy-ybl) / ((xtl-xtr)*(ytl-ybl)); nlt2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr)); nlt2bl[i] = (xx-xbr)*(xy-ytl) / ((xbl-xbr)*(ybl-ytl));</pre>		for(1=0; 1<9; 1++ { fprintf(frez, "
Switch(1)		fprintf(frez,
<pre>case 1: case 8: xx = xt1 - 2.000 * ecx2[i] * delta_t / delta_x; xy = yt1 - 2.000 * ecy2[i] * delta_t / delta_x; break; case 4:</pre>		switch (key_init)
<pre>case 5: xx = xbl - 2.000 * ecx2[i] * delta_t / delta_x; xy = ybl - 2.000 * ecy2[i] * delta_t / delta_x; break; default: break; }</pre>		Cose U: /* nomc for (j=0; j< for (i=0; j
nsl2tl[i] = (1.000 + xx*xtl)*(1.000 + xy*ytl) / 4.000; nsl2tr[i] = (1.000 + xx*xtr)*(1.000 + xy*ytr) / 4.000; nsl2br[i] = (1.000 + xx*xxxx)*(1.000 + xy*ytr) / 4.000; nsl2br[i] = (1.000 + xx*xxx)*(1.000 + xy*yxx) / 4.000;		£10[k] £20[k]
<pre>nll2tr[i] = (xx-xtl)*(xy-ybr) / ((xtr-xtl)*(ytr-ybr)); nll2tl[i] = (xx-xtr)*(xy-ybl) / ((xt-xtr)*(ytl-ybl)); nll2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr)); nll2bl[i] = (xx-xbr)*(xy-ytr) / ((xbl-xbr)*(ybl-ytr));</pre>		break; case 1: /* two k=-1; for (j=0; j<
<pre>switch(i) {</pre>		for (1=0;
<pre>case 2: case 7: xx = xtr - 2.000 * ecx2[i] * delta_t / delta_X; xy = ytr - 2.000 * ecy2[i] * delta_t / delta_x; break; case 3.</pre>		X () () () () () () () () () (
<pre>case 6: xx = xbr - 2.000 * ecx2[i] * delta_t / delta_x; xy = ybr - 2.000 * ecy2[i] * delta_t / delta_x; hreak;</pre>		else

<pre>default: break; }</pre>
<pre>nsr2t1[i] = (1.000 + xx*xt1)*(1.000 + xy*yt1) / 4.000; nsr2tr[i] = (1.000 + xx*xtx)*(1.000 + xy*ytx) / 4.000; nsr2b1[i] = (1.000 + xx*xb1)*(1.000 + xy*yb1) / 4.000; nsr2br[i] = (1.000 + xx*xbx)*(1.000 + xy*ybx) / 4.000;</pre>
<pre>nlr2tr[i] = (xx-xtl)*(xy-ybr) / ((xt-xtl)*(ytr-ybr)); nlr2tl[i] = (xx-xtr)*(xy-ypl) / ((xtl-xtr)*(ytl-ybl)); nlr2br[i] = (xx-xbl)*(xy-ytr) / ((xbr-xbl)*(ybr-ytr)); nlr2bl[i] = (xx-xbr)*(xy-ytr) / ((xbl-xbr)*(ybl-ytr));</pre>
<pre>void init_arrays_nine_square(void) </pre>
FILE *fsav, *frez; int i.j.k, key-rho; double rand.coef = 0.0000000001, dummy; double radius, xc, yc, x,y;
<pre>frez=fopen(rez_name,"a");</pre>
for(1=0; 1<9; 1++)
<pre>fprintf(frez,"1,ecx[i],ecy[i])n%3d %lf %lf\n", i.ecx[i].ecy[i]); fprintf(frez,"i,ecx1[i].ecy1[i]\n%3d %lf %lf\n", fprintf(frez,"i,ecx2[i].ecy2[i]\n%3d %lf %lf\n",</pre>
switch (key_init)
case 0: /* homogeneous binary system */ $k=-1;$ for ($j=0;$ $j<$ nnodes_y; $j++$)
{ for (i=0; i < nnodes_x; i++)
<pre>f k++; f10(k] = nzero1 * (1.000 + rand_coef * f10(k) = (double) rand() / (double) RAND_MAX) - 0.5)); f20(k) = nzero2 * (1.000 + rand_coef *</pre>
} break; case 1: /* two horizontal layers */
$K=-1;$ for (j=0; j< nnodes_y; j++)
for (1=0; i < nnodes_x; i++)
k++; if(j <nnodes_y 2)<="" td=""></nnodes_y>
<pre>f10[k] = nzero1 * (1.000 + rand_coef *</pre>
else {

Jun 25 1999 13:31 wet9init.c	Page 17
<pre>f10[k] = nzero2 * (1.000 + rand_coef *</pre>	
case 2: /* carruston coupte "/ k=-1; for (j=0; j< nnodes_y; j++)	
{ for (i=0; i < nnodes_x; i++)	
k++; if(<=nnodes_x/2)	
f10[k] = nzerolleft * (1. +rand_coef * (((double) rand() / (double) RAND_MAX) - 0.5)); f20[k] = nzerozleft * (1. +rand_coef * * (double) rand() / (double) RAND_MAX) - 0.5));	
else { f10[k] = nzerolright * (1.0 +rand_coef *	
break; case 3: /* circular drop */ radius = delta_x * (idouble) nnodes_y) / 4.000; xc = delta_x * (idouble) nnodes_x) / 2.000; yc = delta_x * (idouble) nnodes_y) / 2.000; nzerol = nzerolleft; nzeroz = nzerozleft;	
$K = -1;$ $for(j=0; j < nnodes_y; j++)$	
<pre>{ y = delta_x * ((double) j); for(i=0; i<nnodes_x; i++)<="" pre=""></nnodes_x;></pre>	
x = delta_x * ((double) 1);	
<pre>1f (x-xc) * (x-xc) + (y-yc) * (y-yc) <= 1f (radius+3.0*delta_x) * (radius+3.0*delta_x)) /* adius+3.0*delta_x) * (radius+3.0*delta_x))</pre>	
radius*radius) */	
key_rho = 4;	
<pre>tsev_rho = 5; if((x-xc)*(x-xc)+(y-yc)*(y-yc)<= (radius+2,0*delta_x)*(radius+2.0*delta_x))</pre>	
<pre>key_zno = 5; if((x-xc)*(x-xc)*(y-yc)*(y-yc)<= if((x-xc)*(x-xc)*(radius+delta_x)*(radius+delta_x))</pre>	
<pre>ksy_rho = 2; if((x-xc)*(x-xc)*(y-yc)*(y-yc)<=radius*radius) ksy_rho = 1; switch(ksy_rho)</pre>	
* (1.000 + rand_coef *	
(((double) rand() / (double) rany_ran, = nzero2right * (1.000 + rand_coef * (((double) rand() / (double) RaND_MAX)	
case 2:	

Jun 25 1999 13:31	wet9init.c Page 18
= nzerolleft ((double = nzerolleft (((double	* (1.000 + rand_coef *
<pre>break; case 3: f10[k] = nzerolleft * (1 f20[k] = nzerolleft * (1 f20[k] = nzerolleft * (1</pre>	* (1.000 + rand_coef * rand() / (double) RAND_MAX) - 0.5))/2.00; * (1.000 + rand_coef * rand() / (double) RAND_MAX) - 0.5))/2.00;
.000 * nze: ((double) .000 * nze:	* (1.000 + rand_coef * / (double) RAND_MAX) - 0.5))/4 * (1.000 + rand_coef * / (double) RAND_MAX) - 0.5))/4 / (double) RAND_MAX) - 0.5))/4
zerolright ((double) zero2left'	00 + rand_coef * / (double) RAND_MAX) - 0 / (double) RAND_MAX) - 0 / (double) RAND_MAX) - 0
• •	
<pre>break; case 4: /* circular bubble */</pre>	
case 5: /* sessile drop */	
Dreak; case 6: /* pendant drop */ hreak:	
case 7: /* sessile bubble */	
case, rendant bubble */	
case 9: /* liquid bridge */	
for ($j=0$; $j<$ nnodes_y; $j++$)	
for (i=0; i < nnodes_x; i++)	
k++; if((i <nnodes_x (i="" 4)="" ="">3*</nnodes_x>	(1>3*nnodes_x/4))
f10[k] = 0.000+0; f20[k] = nzero2 * (1.000 (((double) ran	000 + rand_coef * rand() / (double) RAND_MAX) - 0.5));
} else s	
f10[k] = nzerol * (1.000 + 1) ((double) rand() f20[k] = 0.000e+0;	0 + rand_coef *. nd() / (double) RaND_MAX) - 0.5));
•	
case 10: /* gas bridge */ k=-1; for (i=0; i< nnodes v; i++)	
-0; 1 < nnodes	
k++; if((i <nnodes_x 3) (i="">2* {</nnodes_x>	(1>2*nnodes_x/3))

8

Page 20	
D.	
/*	
all */ right wall	
wet9init.c	
_x] = 3; /odes_x-1] =	
de[j*nnodes le[(j+1)*nn	
<pre>99 13:31</pre>	:-
) break;)

6

Jun 3 1999 12:55	wet9draw.c	Page 1
/*************************************	*** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** **	
<pre>finclude <stdio.h> #include <math.h> #include <stdlib.h> #include <stdlib.h> #include <string.h></string.h></stdlib.h></stdlib.h></math.h></stdio.h></pre>		
clude "wet9head.h"		
<pre>/* you may modify gray_levels */ int gray_levels = 63, base = 3</pre>	35,	
<pre>void xv_new(double n0[], double n2[], double double n5[], double n6[], double char arg_name[], double maxval)</pre>	n2[], double n3[], double n4[], n6[], double n7[], double n8[],	
<pre>int i, val; char xv_name[128]; FILE *fxv;</pre>		
<pre>sprintf(xv_name, "%s.%06d.xpm", arg_name, iter); fxv = fopen(xv_name, "w"); fprintf(fxv, "/* XPM */\nstalic char * init_xpm[] fprintf(fxv, "("%d %d %d %d", nnodes_x, nnodes_y, for(i = 0; i < gray_levels; i++) fprintf(fxv, "\", \n\"%c\tg gray&d", base + i, fprintf(fxv, "\", \n\"%c\tg gray&d", base + i, filt(fxv, "\", \n\"%c\tg gray&d", base + i,</pre>	<pre>, arg_name, iter); c char * init_xpm[] = {\n^*); nnodes_x, nnodes_y, gray_levels + 1, 1); ++) ray_levels);</pre>	
/* for(i = 0; i <= nnodes_all; i++)	/* lattice -> xpm */ +)	
<pre>fit(1 == ((1+1) % nnodes_X)) fprintf(fxv,"\",\n\""); val = floor((n0[ii] + n1[ii]) if(val > gray_levels) val = gray_levels; if(val < 0)</pre>) + n2[i] + n3[i] + n4[i] + n5[i] + + n8[i]) * gray_levels / maxval);	
val = 0; fprintf(fxv,"%c", base + val)	al);	
<pre>fprintf(fxv,"\");\n"); folose(fxv); }</pre>		
<pre>void xv(double n0[], double n1[], double n5[], double n5[], char arg_name[])</pre>	, double n3[], double n4[], , double n7[], double n8[],	
<pre>'FILE *fxv; char xv_name[128]; int i,k_val; int i,k_val; sprintf(xv_name,"%s,%05d",arg_name,iter); fxv = fopen(xv_name,"wt"); fprintf(fxv,"P2\n%3d%4d\n63\n",nnodes_x,nnodes_y) i=0; for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>	name,iter); ,nnodes_x,nnodes_y);	
<pre>{ i+; val = floor((n0[k]+n1[k]+n</pre>	floor((n0[k]+n1[k]+n2[k]+n3[k]+n4[k]+n5[k]+ n6[k]+n7[k]+n8[k])*63.0); L > 63)	

Jun 3 1999 12:55 wet9draw.c	Page 2
<pre>val = 63; if(val < 0) val = 0; val = 0; /* val = 63 - val; */ fprint(fxv,*&3d",val); if(i == 15)</pre>	
1=0; fprintf(fxv,"\n");	
<pre>if(1) fprintf(fxv,"\n"); fclose(fxv); }</pre>	
void wet9_drop_profile(void)	
FILE *pnlx, *pn2x;	
<pre>char nlx_profile_name[128]; char n2x_profile_name[128];</pre>	
<pre>int i, j, k; double nloc1, nloc2; double nlx[nnodes_x], n2x[nnodes_x];</pre>	
<pre>sprintf(nlx_profile_name, "PRESSIX*s.*806d",id_name,iter); sprintf(n2x_profile_name, "PRESSZX*s.*806d",id_name,iter);</pre>	
<pre>pnlx = fopen(nlx_profile_name,"wt"); pn2x = fopen(n2x_profile_name,"wt");</pre>	
<pre>k = nnodes_x * nnodes_y/2 - 1; for(1=0; 1<nnodes_x; 1++)<="" pre=""></nnodes_x;></pre>	
k ++; nloc1 = f10(k)+f11(k)+f12(k)+f13(k)+f14(k)+f15(k)+f16(k)+ f17(k)+f18(k); nloc2 = f20(k)+f21(k)+f22(k)+f23(k)+f24(k)+f25(k)+f26(k)+ f27(k)+f28(k);	
<pre>fprintf(pnlx,"%d %25.201f(\n",i,nloc1); fprintf(pn2x,"%d %25.201f(\n",i,nloc2); }</pre>	
fclose(pnlx); fclose(pn2x); }	
yold wet9_channel_profile(void)	
FILE *pn1, *pn2, *pntot, *prhotot, *pvx, *pvx1, *pvx2, *pgvx, *pvisco;	
FILE *pf0, *pf1, *pf2, *pf3, *pf4, *pf5, *pf6, *pf7, *pf8;	
char nl_profile_name[128]; char nl_profile_name[128]; char ntot_profile_name[128]; char rhotot_profile_name[128]; char vx_profile_name[128]; char vx_profile_name[128]; char vx_profile_name[128]; char vx_profile_name[128]; char vx_profile_name[128]; char vx_profile_name[128];	
<pre>char f0_name[128]; char f1_name[128]; char f2_name[128];</pre>	

Printed by sofonea from flumag3.mec.utt.ro

wet9draw.c

wet9draw.c

5
Ħ
ē
~. -:
ğ
Ha
₫
_
5
=
g
5
ğ
S
ô
g
Ĕ
Ų.

wet9draw.c

Jun 3 1999 12:55	wet9draw.c	က	Jun 3 1999 12:55
char f3_name[128]; char f4_name[128]; char f5_name[128]; char f6_name[128]; char f7_name[128]; char f8_name[128];			
int i,j,k; double nloc1, nloc2, nl double vx[nnodes_y], y[double vx[nnodes_y], double nl[nnodes_y], double nlsnodes_y], nl	int i,j,k; double nloc1, nloc2, nloc, rho1, rho2, rholoc, uloc1, uloc2, uloc, mean_mass; double vx[nnodes_y], y[nnodes_y], grad_vx[nnodes_y]; double vx[nnodes_y], vx2[nnodes_y]; double nl[nnodes_y], n2[nnodes_y], ntot[nnodes_y]; double nl[nnodes_y], n2[nnodes_y], rhotot[nnodes_y]; double visco, ivisco, suml, sum2, channel_width;		
sprintf (nl_profile_name sprintf (n2_profile_name sprintf (ntot_profile_na sprintf (rwtprofile_name sprintf (vx_profile_name sprintf (vx_profile_name sprintf (vx_profile_name sprintf (vx_profile_name sprintf (visco_name, "FVI	<pre>sprintf(nl_profile_name, "PN1%s.%06d",id_name,iter); sprintf(n2_profile_name, "PN2%s.%06d",id_name,iter); sprintf(ntot_profile_name, "PN07%s.%06d",id_name,iter); sprintf(rw.profile_name,"PNX%s.%06d",id_name,iter); sprintf(vx_profile_name,"PVX%s.%06d",id_name,iter); sprintf(vx_profile_name,"PVX%s.%06d",id_name,iter); sprintf(vx_profile_name,"PVX%s.%06d",id_name,iter); sprintf(vx_profile_name,"PVX%s.%06d",id_name,iter); sprintf(visco_name,"PVXSco.%06d",id_name,iter);</pre>		else block uloc uloc uloc
<pre>sprintf(f0_name, "PFD%s.%06d" id_name, iter); sprintf(f1_name, "PF1%s.%06d", id_name, iter); sprintf(f2_name, "PF2%s.%06d", id_name, iter); sprintf(f3_name, "PF3%s.%06d", id_name, iter); sprintf(f4_name, "PF4%s.%06d", id_name, iter); sprintf(f6_name, "PF6%s.%06d", id_name, iter); sprintf(f6_name, "PF6%s.%06d", id_name, iter); sprintf(f7_name, "PF6%s.%06d", id_name, iter); sprintf(f7_name, "PF6%s.%06d", id_name, iter);</pre>	*06d",id_name,iter); .%06d",id_name,iter);		else if (rh if (rh))))))))))))))))))))))))))))))))))))
pf0 = fopen(f0_name, "wt"); pf1 = fopen(f1_name, "wt"); pf2 = fopen(f2_name, "wt"); pf3 = fopen(f3_name, "wt"); pf4 = fopen(f4_name, "wt"); pf5 = fopen(f4_name, "wt"); pf6 = fopen(f6_name, "wt"); pf7 = fopen(f6_name, "wt"); pf7 = fopen(f6_name, "wt"); pf8 = fopen(f8_name, "wt");	() ; () ;		case 2:
<pre>pnl = fopen(nl_profile_name, "wt"); pn2 = fopen(nl_profile_name, "wt"); pntot = fopen(ntot_profile_name, "wt"); prv = fopen(ntot_profile_name, "wt"); pvx = fopen(vx_profile_name, "wt"); pvx1 = fopen(vx_profile_name, "wt"); pvx2 = fopen(vx_profile_name, "wt"); pvx2 = fopen(vx_profile_name, "wt"); pvx2 = fopen(vx_profile_name, "wt"); pvxc = fopen(visco_name, "wt"); pvisco = fopen(visco_name, "wt"); else pvisco = fopen(visco_name, "wt");</pre>	name,"wt");lane,"wt"); file_name,"wt");profile_name,"wt");le_name,"wt");le_name,"wt");profile_name,"wt");laname,"wt");laname,"wt");laname,"wt");laname,"wt");laname,"wt");laname,"wt");		vaz[j] += vaz[j] += n j
k = 0; for(j=0; j <nnodes_y; j++)<="" td=""><td>(++)</td><td></td><td>fprintf(pnl,' fprintf(pn2,' fprintf(pn2,'</td></nnodes_y;>	(++)		fprintf(pnl,' fprintf(pn2,' fprintf(pn2,'
y(j) = ((double) j) vx[j] = 0.0000; vx1[j] = 0.0000; vx2[j] = 0.0000;	j) * delta_y;		fprint (prior fprior fp
n1(1) = 0.0000; n2(1) = 0.0000; ntot(1) = 0.0000; rhotot(1) = 0.0000; for(1=0; i <nnodes_x; 1++)<="" td=""><td>); X; 1++)</td><td></td><td>fprint (pf0," fprint (pf1," fprint (pf2," fprint (pf3,"</td></nnodes_x;>); X; 1++)		fprint (pf0," fprint (pf1," fprint (pf2," fprint (pf3,"

	<pre>rhotot(i) += rholoc; vx[i] += uloc; vx[i] += uloc; vx[i] += uloc; vx[i] += uloc; vx2[i] += uloc; vx2[i] += uloc; vx2[i] += (double) nnodes_x); nf(i] /= ((double) nnodes_x); nf(i] /= ((double) nnodes_x); rhotot(i] /= ((double) nnodes_x); vx[i] /= ((double) nnodes_x); vx[i] /= ((double) nnodes_x); vx[i] /= ((double) nnodes_x); vx2[i] /= ((double) nnodes_x); fprintf(pn1,**lf \$25.201f\n",y[i],nl[i]); fprintf(pncot,**lf \$25.201f\n",y[i],not([i]); fprintf(pvx,**lf \$25.201f\n",y[i],vx1[i]); fprintf(pvx,**lf \$25.201f\n",y[i],vx1[i]); fprintf(pfv,**lf \$25.201f\n",y[i],vx1[i]); fp</pre>
--	--

Jun 3 1999 12:55 wet9draw.c	Page 5
<pre>fprintf(pf4,"%lf %25.20lf(\n",y(j)fl4(j*nnodes_x]); fprintf(pf5,"%lf %25.20lf\n",y(j)fl5(j*nnodes_x]); fprintf(pf6,"%lf %25.20lf\n",y(j)fl6(j*nnodes_x]); fprintf(pf7,"%lf %25.20lf\n",y(j)fl7(j*nnodes_x]); fprintf(pf8,"%lf %25.20lf\n",y(j)fl8(j*nnodes_x]); }</pre>	
<pre>grad_vx[0] = (vx[1]-vx[0])/delta_y; fprintf(pgvx,"%lf %25.201f\n",y[0],grad_vx[0]); for(j=1; j<nnodes_y-1; j++)<="" pre=""></nnodes_y-1;></pre>	
<pre>' grad_vx[j] = (vx[j+1]-vx[j-1])/(2.0000*dalta_y); fprintf(pgvx,"%lf %25.201f\n",y[j],grad_vx[j]);</pre>	
<pre>grad_vx[nnodes_y-1] = (vx[nnodes_y-1]-vx[nnodes_y-2])/delta_y; fprintf(pgvx,"%lf %25.201f\n",y[nnodes_y-1],grad_vx[nnodes_y-1]);</pre>	
<pre>channel_width = ((double) (nnodes_y-1)) * delta_y; suml = sum2 = 0.00000; for(j=0; j<nnodes_y; j++)<="" pre=""></nnodes_y;></pre>	
<pre>{ sum1 += (y[j]*channel_width - y[j]*y[j]) * (y[j]*channel_width - y[j]*y[j]); sum2 += vx[j] * (y[j]*channel_width - y[j]*y[j]);</pre>	
<pre>sum2 *= 2.0000; mean_mass = (mass1*nzero1 + mass2*nzero2) / (nzero1+nzero2);</pre>	
<pre>/* if(vx[nnodes_y/2]) printf("%lf %lf %e %e %e\n",mean_mass,(force_x*channel_width*channel_width)</pre>	width)/
else printf("%lf\n",mean_mass);	
<pre>if(sum2) if(sum2) visco = (force_x / (mean_mass * amu)) * sum1 / sum2; else</pre>	
<pre>visco = 0.0000; fprintf(pvisco, "%8.8d %e\n",iter,visco);</pre>	
fclose(pnl); fclose(pnt); fclose(pntot); fclose(pxhotot); fclose(pxxl); fclose(pxxl); fclose(pxxl); fclose(pxxl); fclose(pxxl); fclose(pxxl);	
fclose(pf0); fclose(pf1);	
fclose(pf3); fclose(pf3); fclose(pf4);	
fclose(pf5); fclose(pf6); fclose(pf7); fclose(pf8);	
void wet9_couple_profile(void)	
<pre>! FILE *pntct, *prhotot, *prho1, *prho2, *pomegal, *pomeganl, *pomega2, *pomegan2, *pj2, *pu1, *pu2, *pu, *pu2bis;</pre>	
<pre>char ntot_profile_name[128]; char rhotot_profile_name[128]; char rho1_profile_name[128];</pre>	

Jun 3 1999 12:55	wet9draw.c	Page 6
char rho2_profile_name[128]; char omegal_profile_name[128]; char omegal_profile_name[128]; char omega2_profile_name[128]; char omega2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2profile_name[128]; char u2prs_profile_name[128]; char u2prs_profile_name[128];	88]; 88]; 128]; 18]; 18];	
int i,j,k,kk,nval; double nloc1, nloc2, rholoc1, rholoc2, rholoc, omega2loc, omegan2loc, uloc2li, uloc2, jloc2, ulc double uloc1bis, uloc2bis, ulocbis; double rhol, rho2, rho, omegal, omegan1, omega2, ul, u2, j2, u, u2;is, ubis; double ntot, rhotot;	11, rholoc2, rholoc, omegalloc, omeganlloc, loc1, uloc2, jloc2, uloc3, ulocbis; egal, omeganl, omegal, omeganl, segal,	
sprint (ntot_profile_name,") sprint (rhot_profile_name,") sprint (rhol_profile_name,") sprint (omegal_profile_name sprint (omegal_profile_name sprint (omegal_profile_name sprint (omegal_profile_name sprint (omegal_profile_name sprint (ul_profile_name,") sprint (ul_profile_name,") sprint (ul_profile_name,") sprint (ul_profile_name,") sprint (ul_profile_name,") sprint (ul_profile_name,") sprint (ul_bis_profile_name,") sprint (ul_bis_profile_name,") sprint (ul_bis_profile_name,") sprint (ul_bis_profile_name,") sprint (ul_bis_profile_name,")	<pre>sprintf(ntot_profile_name, "PNTOTAL%s.%06d",id_name,iter); sprintf(thot_profile_name,"PRHOT%s.%06d",id_name,iter); sprintf(thot_profile_name,"PRHOT%s.%06d",id_name,iter); sprintf(thot_profile_name,"PRHOT%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(ul_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(ul_profile_name,"PUT%s.%06d",id_name,iter); sprintf(ul_profile_name,"PUT%s.%06d",id_name,iter); sprintf(ul_profile_name,"PUT%s.%06d",id_name,iter); sprintf(ul_profile_name,"PUTSS1%s.%06d",id_name,iter); sprintf(ul_profile_name,"PUTSIS%s.%06d",id_name,iter); sprintf(ul_brofile_name,"PUTSIS%s.%06d",id_name,iter);</pre>	
<pre>pntot = fopen(ntotprofile_name,"wt"); prhtot = fopen(thotot_profile_name,"wt"); prhtol = fopen(thotot_profile_name,"wt"); prhtol = fopen(thoto_profile_name,"wt"); pomegal = fopen(omegal_profile_name,"wt"); pomegal = fopen(omegal_profile_name,"wt"); pomegal = fopen(omegal_profile_name,"wt"); pomegal = fopen(omegal_profile_name,"wt"); pj2 = fopen(j2_profile_name,"wt"); pu1 = fopen(u2_profile_name,"wt"); pu2 = fopen(u2_profile_name,"wt"); pu3 = fopen(u2_profile_name,"wt"); pu4 = fopen(u2_profile_name,"wt");</pre>	name, "wt"; file_name, "wt"; name, "wt";name, "wt";name, "wt"; file_name, "wt"; fil	,
<pre>pubbis = fopen(u2bis_profile_name,"wt") pubis = fopen(ubis_profile_name,"wt");</pre>	le_name,"wt"); _name,"wt");	
ntot = 0.0000; rhotot = 0.0000; rho1 = 0.0000; rho2 = 0.0000; omegal = 0.0000; omegal = 0.0000; omegal = 0.0000; ul = 0.0000;		

Printed by sofonea from flumag3.mec.utt.ro

wet9draw.c

mag3.mec.utt.ro	Page 8
Printed by sofonea from flumag3.mec.utt.ro	wet9draw.c

Jun 3 1999 12:55 wet9draw.c	Page 7	Jun 3 1999 12:55
k = i + j * nnodeg x; nloc1 = £10[k]+£11[k]+£12[k]+£13[k]+£14[k]+£15[k]+£16[k]+ £17[k]+£18[k]; nloc2 = £20[k]+£2[k]+£22[k]+£24[k]+£25[k]+£26[k]+ £27[k]+£28[k];) /* /* if(nloc1) uloclbis
<pre>/* printf("k=%d nloc1=%g nloc2=%g\n",k,nloc1,nloc2);</pre>		else ulocibié if (nloc2) uloc2bié
rnolog = rnolog1 + rnolog2; /* printf("k=\$d rholog1=\$g rholog2=\$g rholog=\$g\n",		else uloc2bis ulocbis =
<pre>cmegalloc = rholoc1 / rholoc; cmegalloc = rholoc2 / rholoc; }</pre>		
<pre>else</pre>		Thotot += Thotot += Thotot += Thot += Thot += Thotot +
<pre>if(nloc1+nloc2) (ameganloc = nloc1 / (nloc1+nloc2); comegan2loc = nloc2 / (nloc1+nloc2); else else</pre>		or t = 10 u = 10 u + = uloo u + = uloo u + z = uloo u = uloo u = uloo u = uloo u = uloo
{ omegan11cc = 0.0000; omegan2lcc = 0.0000; }		
<pre>if(nloc1) uloc1 = (fil[k]*ecx1[1]+f12[k]*ecx1[2]+f13[k]*ecx1[3]+</pre>		omegal /= (1doubl omegal /= (1d omegal /= (1d omeganl /= (1d omeganl /= (1doubl)2 /= (1doubl ul /= (1doubl
<pre>if(nloc2) uloc2 = (f21[k]*ecx2[1]+f22[k]*ecx2[2]+f23[k]*ecx2[3]+</pre>		u /= ((double u /= ((double ubis /= ((double ubis /= ((dou fprintf(prhot fprintf(prhot fprintf(prhot
/* printf("k=%d uloc1=%g uloc2=%g\n",k,uloc1,uloc2); */		fprintf(prho2 fprintf(pomeg fprintf(pomeg fprintf(pomeg fprintf(pomeg
<pre>if(rholoc)</pre>		fprintf(pj2, m fprintf(pj1, m fprintf(pj2, m fprintf(pj2, me fprintf(pj2, me
uloc = 0.0000; /* printf("k=%d rholoc=%g\n",k,rholoc); */		fprintf(pubis */ fclose(pntot);

/* if(nloc1) uloc1bis = (uloc1 * (1.000 - 0.500*delta_t/taul) + uloc1bis = (uloc1 * (1.000 + 0.500*delta_t/taul) +	
else ulocibis = 0.000; if(nloc2) uloc2bis = (uloc2 * (1.000 - 0.500*delta_t/tau2) + uloc*0.500*delta_t/tau2)/nloc2;	
<pre>else uloc2bis = 0.000; uloc2bis = (rholoc1*uloc1*(1.000 - 0.500*delta_t/tau1) +</pre>	
<pre>ntot += (nloc1 + nloc2); rnotot += (rholoc1 + rholoc2); rnot += rholoc2; and += rholoc2; u2 += uloc2; u3 += uloc2; u4 += uloc2; u5 += uloc2); u5 += uloc2); u6 += uloc2; u7 += uloc2; u8 += uloc2; u8 += uloc2; u9 += uloc2; u1 += uloc2; u2 += uloc2; u3 += uloc2</pre>	·
} fclose(pntot);	

wet9draw.c

_
5
₹
ပ္ပဲ
Ĕ
က်
ag
Ε
₽
Ε
6
a
ē
₽
တ္တ
\geq
5
ĕ
₽.
ũ.

Jun 3 1999 12:55	wet9draw.c Page	ge 9	L	Jul
fclose(prhotot); fclose(prhol); fclose(prhol); fclose(prhol); fclose(pomegal); fclose(pomegal); fclose(pomegal); fclose(pomegal); fclose(pomegal); fclose(pomegal); fclose(pul);				
<pre>void test_ntot (void) EIIE *ptot; int k; double ntot1=0.0000, ntot2=0.0000; char ntot_name(128); sprintf(ntot_name(128); ptot = fopen(ntot_name, "aw"); for (k=0; k<nnodes_all; "iter="%d" *p="" *pux,="" *puy,="" *pxc,="" *pxn,="" *pyc,="" +="f20[k]+f21[k]+f22[k]+f23" double="" file="" fprintf(ptot,="" iter,ntot1,ntot2);="" k++)="" nloc1,="" nloc2;<="" ntot1="%25.201" ntot2="" pre="" quiver(void)="" void="" }=""></nnodes_all;></pre>	<pre>set_ntot(void) ; *ptot; la ntotl=0.0000, ntot2=0.0000; la ntotl=0.0000, ntot2=0.0000; intot.name[128]</pre>			
<pre>double uxl, uyl, ux2, uy2, ux, uy, xc, yc; thi 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,</pre>	x1, uy1, ux2, uy2, ux, uy, xc, yc;			~
uxl = (fl1 K)*ecx f14[k]*ecx uyl = (fl1[k]*ecy	<pre>[III]k,*ecx[1]+II2 k,*ecx[2]+II3 k,*ecx[6]) / nloc1; f14(k)*ecx[4]+f15(k)*ecx[5]+f16(k)*ecx[6]) / nloc1; (f11(k)*ecy[1]+f12[k)*ecy[2]+f13(k)*ecy[3]+</pre>			- 1

Jun 3 1999 12:55	s wet9draw.c	Page 10
} else	f14[k]*ecy[4]+f15[k]*ecy[5]+f16[k]*ecy[6]) / nloc1;	
ux1 uy1 ly1	1 = 0.0000; 1 = 0.0000;	
11 (nlocz) { ux2 = uy2 =	<pre>c2) 2 = (f21[k]*ecx[1]+f22[k]*ecx[2]+f23[k]*ecx[3]+</pre>	
else { ux2 uy2	11 11	
x n n n (t	<pre>(mass1 * nloc1 * ux1 + mass2 * nloc2 * ux2) / (mass1 * nloc1 + mass2 * nloc2); (mass1 * nloc1 + uy1 + mass2 * nloc2 * uy2) / (mass1 * nloc1 + mass2 * nloc2); delta_x * ((double) 1); delta_y * ((double) 1); 1;</pre>	
. int	f(pux, " %lf",ux); f(pxc, " %lf",uy); f(pxc, " %lf",xc); f(pxc, " %lf",yc); f(pxn, " %5d",yn); f(pyn, " %5d",yn);	
<pre>fprint(pux,"\n"); fprint(puy,"\n"); fprint(pxc,"\n"); fprint(pyc,"\n"); fprint(pxn,"\n"); fprint(pxn,"\n"); fprint(pxn,"\n");</pre>	x "\n"); y, "\n"); c, "\n"); i, "\n"); i, "\n"); i, "\n");	
fclose (pux); fclose (puy); fclose (pxc); fclose (pxn); fclose (pxn);		

.c Page 11		
Jun 3 1999 12:55 wet9draw.c		

Printed by sofonea from flumag3.mec.utt.ro

wet9cs.c

lumag3.mec.utt.ro	Page 2
Printed by sofonea from flur	wet9cs.c

*************************************	fs2[8] = (nf28[nvp
** Wet9cs.c	E
《水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水水	dummy = massl
	S S S S S S S S S S S S S S S S S S S
<pre>#include <stdlib.n> #include <math.h></math.h></stdlib.n></pre>	
<pre>#include "wet9head.h"</pre>	/caul +
double compute_centered_sources_bulk(int sigma, int index, int k,	/tau2)
int nvpius, int nvminus, double nf10[], double nf11[], double nf12[], double nf13[], double nf14[], double nf15[],	uy = (massl * /taul +
<pre>double nf16[], double nf17[], double nf18[], double nf20[], double nf24[], double nf22[], double nf23[], double nf24[], double nf25[], double nf25[], double nf24[], double nf28[])</pre>	mass2 * (fs2[fs2] /tau2) / dumm
double dummy, ux, uy, uu, neq, cfl, source; double fsl[9], fs2[9];	<pre>uu = ux*ux + uy*uy; switch(sigma) {</pre>
switch(sigma)	case 1: dummy = (ecx1[i
case 1:	/* printf("dummy=% */
break;	neg = ww[index]
ofl = cfl2; break;	(1,000 + thre three_over_t
fe1[0] = (nf10[nvm]ns1 + nf10[nvminns]) / 2.000 -	source = - cta (ecprod1[ind
* (nf11[nvplus] + nf11[nvminus]); 2.000	printf("k=%d fs1+source=%25.20e
1 +	l[index]);
* (nf12[nvplus] - = (nf13[nvplus] +	*/ break;
* (nf13[nvplus] - = (nf14[nvplus] +	case 2: dummy = (ecx2[i
* (nf14[nvplus] - nf14 = (nf15[nvplus] + nf15	neq = ww[index]
* (nf15[nvplus] - = (nf16[nvplus] +	(1.000 + thre three_over_t
* (nf16[nvplus] - nf16[nvminus]); = (nf17[nvplus] + nf17[nvminus]) /	source = - ctau (ecprod2[ind
* (nf17[nvplus] - nf17[nvminus]); = (nf18[nvplus] + nf18[nvminus]) /	break;
* (nfl8[nvplus] - nfl8[nvminus]);	return source;
= (nf20[nvplus] + nf20[nvminus] * (nf20[nvminus] - nf20[nvminus]	double compute center
= (nf21[nvplus] + nf21[nvminus]) * (nf21[nvplus] - nf21[nvminus])	
= (nf22[nvplus] + nf22[nvmlnus] * (nf22[nvmlnus]	
= (nf23[nvplus] + nf23[nvminus] * (nf23[nvminus]	
= (nf24[nvplus] + nf24[nvminus] * (nf2/form] - nf2/forminus]	
= (nic*{nvplus} nic*{invminus}) = (files (nvminus) nic*{invminus}) * (files (nvminus) nic*{invminus})	
= (nf25(nvplus) + nf25(nvmlnus))	double fal[9],
1 + 1	switch(sigma)

fs2[8] = (nf28[nvplus] + nf28[nvminus]) / 2.000 - cf1 * (nf28[nvplus] - nf28[nvminus]);
<pre>dummy = mass1 * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+</pre>
<pre>uu = ux*ux + uy*uy; switch(sigma) data dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12; printf(*dummy=%25.20e\n",dummy);</pre>
*/ neq = ww[index] * [fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[5]+fs1[6]+ fs1[7]+fs1[8]) * (1.000 + three * dummy + nine_over_two * dummy - three_over_two * uv cspecd[2]; source = - ctaul * (fs1[index] - neq) + (ecpred1[index] - csforce_x * ux - csforce_y * uy) * neq;
<pre>printf("k=%d index=%d fs1=%25.20e neq=%25.20e source=%25.20e fs1-neq=%25.20e fs1+source=%25.20e ux=%e uy-%e ecprod=%e\n" k,index,fs1[index],neq,source,fs1[index]-neq,fs1[index]+source,ux,uy,ecprod</pre>
<pre>dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22; neq = ww[index] * (fs2[0]+fs2[1]+</pre>
double compute_centered_sources_boundary1(int sigma, int index, int k, int nvplus, int nvminus, double unplus, int nvminus, double un_boundary, double un_boundary, double nf10[], double nf11[], double nf15[], double nf18[], double nf18[], double nf18[], double nf18[], double nf18[], double nf21[], double nf22[], double nf22[], double nf22[], double nf22[], double nf22[], double nf25[], double nf27[], double nf28[]),
<pre>double dummy, uu, neq, cfl, source; double fel[9], fe2[9]; switch(sigma) {</pre>

wet9cs.c

lumag3.mec.utt.ro	Page 4
Printed by sofonea from flur	wet9cs.c

May 25 1999 16:33	wet9cs.c	Page 3	May 25 1999 16:33
1: E1 = cfll; = (nfl0[nvplus] + (nfl0[nvplus] = (nfl1[nvplus] + (nfl1[nvplus]	nf10[nvminus] / 2.000 nf10[nvminus]); 2.000 nf11[nvminus] / 2.000 nf11[nvminus]);		return source;) double compute_center
fs1[2] = (nf12[nvplus] cf1 * (nf12[nvplus] fs1[3] = (nf13[nvplus] cf1 * (nf13[nvplus] fs1[4] = (nf13[nvplus] cf3 * (nf14[nvplus] cf3 * (nf14[nvplus]	+ nf12[nvminus]) / 2.000 - - nf12[nvminus]); + nf13[nvminus]) / 2.000 - - nf13[nvminus]); - nf14[nvminus]) / 2.000 - - nf14[nvminus]) / 2.000 -		
* *	nvminus]) nvminus]) nvminus])	- 0.	{ double dummy, uu, n double fsl[9], fs2[
# # * # # # # # # # # # # # # # # # # #	nvminus]) nvminus]) nvminus])		switch(sigma) (ase 1:
1 = cfl2; = (nf20 = (nf20 = (nf21 = (nf21 = (nf21	nf20[nvminus]) nf20[nvminus]) nf21[nvminus]) nf21[nvminus]) nf22[nvminus])		
cc * (nf22[nvplus fs2[3] = (nf23[nvplus cf * (nf23[nvplus fs2[4] = (nf24[nvplus cf * (nf24[nvplus fs2[5] = (nf25[nvplus cf	- nf22[nvminus]); + nf23[nvminus]) / 2.000 - - nf23[nvminus]) / 2.000 - + nf24[nvminus]) / 2.000 - - nf24[nvminus]) / 2.000 -		
* (nf26 * (nf26 * (nf26 * (nf27 * (nf27			f22[4] = nf24[4] f22[5] = nf25[f22[6] = nf26[f22[7] = nf27[b] f22[8] = nf28[7] break;
Dreak;) uu = ux_boundary*ux_bour	Dreak; } uu = ux_boundary*ux_boundary + uy_boundary*uy_boundary;		<pre>uu = ux_boundary*ux switch(sigma) {</pre>
<pre>switch(sigma) { case 1: dummy = (ecx1[index]</pre>	<pre>ch(sigma) se 1: dummy = (ecx1[index]*ux_boundary + ecy1[index]*uy_boundary) / cspeed12;</pre>		case 1: dummy = (ecx1[: neq = ww[index,
<pre>printf("dummy=%25.20e\n", dummy);</pre>	<pre>De\n", dummy); [531[0] +fs1[1] +fs1[2] +fs1[3] +fs1[4] +fs1[5] +fs1[6] +</pre>		three_over_ source = - cts (ecprod1[in
(1.000 + three * cthree * cthree * source = - ctaul * ceprodi[index] - ctaul * ceprodi[index] - cthree * cthre	(1.000 + three * dummy + nine_over_two * dummy * dummy - three_over_two * uu / cspeedil2; source = - ctaul * (fsl[index] - neq) + (ecprod1[index] - csforce_x*ux_boundary - csforce_y*uy_boundary) * neq;		case 2: dummy = (ecx2[: neq = ww[index // non + thy
case 2: dummy = (ecx2[index] neq = ww[index] * (f	<pre>break; break; dunmuy = (ecx2[index]*ux_boundary + ecy2[index]*uy_boundary) / cspeed22; neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+</pre>		three_over. source = - ctal (ecprod2[1m break;
(1.000 + three * c three_over_two * source = - ctaul * c (ecprod2[index] - break;	<pre>(1.000 + three * dumny + nine_over_two * dumny * dummy - three_over_two * uu / capeed22; curree = - ctau2 * (fs2[index] - neg) +</pre>		return source;

uble compute_centered_sources_boundary2(int double compute_centered_sources_boundary2(int double nfil) double nfild adubte nfild couple nfild double nfild double nfild couple nfild double nfild couple nfild switch(sigma) switch(sigma) case 1: fal[0] = nfil(k] - cfil; (nfilk] - nfill nfil
--

wet9cs.c

May 25 1999 16:33 wet9cs.c	Page 5
<pre>void compute_centered_sources(double nf10[], double nf15[], double nf18[], double nf18[], double nf17[], double nf18[], double nf20[], double nf21[], double nf22[], double nf23[], double nf24[], double nf25[], double nf25[], double nf27[], double nf25[],</pre>	
<pre>int k; for (k=0: k<nnodes all:="" k++)<="" pre=""></nnodes></pre>	
_ * _ j	
catol * (nf28 k] - neq28 k] + - caforce_x * uxloc[k] - csforce_y * uyloc[k]) *	
<pre>void centered_cfl(double cfl, double nf0[], double nf1[], double nf5[], double nf6[], double nf7[], double nf5[], double nf6[], double nf7[], double nf6[], double nff0[], double nf7[], double nff2[], double nff2[], double nff1[], double nff8[], double nff6[], double nff7[], double nff8[], double source0[], double source2[], double source0[], double source2[], double source0[], double source2[], double source0[], double source0[])</pre>	
{ int k;	
for (k=0; k <nnodes_all; k++)="" td="" {<=""><td></td></nnodes_all;>	
Switch (Doundary_mode (A) /	

= nf0[k] + source0[k]; = (nf1[nv1[k])+nf1[nv3[k]])/2.000 - (nf1[nv1[k])-nf1[nv3[k]])/2.000 + source1[k]; = (nf2[nv2[k])+nf2[nv4[k]])/2.000 - -nf2[nv2[k])+nf2[nv4[k]])/2.000 + source2[k].
+ 000 6/
72.000 +
/2.000 + /2.000 / /2.000 / /2.000 / /2.000 / /2.000 / /2.000 / /2.000 / /2.000 / /2.000
(nrs nvs k] -nrs nvs k]])/2.000 + sources k]; = nf0[k] + source0[k];
<pre>(nfi[nv1[k]]-nil[nv3[k]])/2.000 + sourcel[k]; (nfi[nv1[k]]-nfi[nv3[k]])/2.000 + sourcel[k];</pre>
= nf6[k] - cf] * (nf6[nv6[k]]-nf6[k]) + source6[k]; = nf7[k] + cf] * (nf7[nv5[k]]-nf7[k]) + source7[k]; = nf8[k] + cf] * (nf8[nv6[k]]-nf8[k]) + source8[k];
= nf0[k] + source0[k]; = (nf1[nv1[k]]+nf1[nv3[k]])/2.000 -
<pre>(nf1[nv1[k]]-nf1[nv3[k]])/2,000 + source1[k]; = nf2[k] + cf1 * (nf2[nv4[k]]-nf2[k]) + source2[k]; = (nf3[nv3[k]]+nf3[nv1[k]])/2,000</pre>
<pre>(nf3[nv3[k]]-nf3[nv1[k]])/2.000 + source3[k]; = nf4[k] - cf1 * (nf4[nv4[k]]-nf4[k]) + source4[k]; = nf5[k] + cf1 * (nf6[nv8[k]]-nf5[k]) + source5[k]; = nf6[k] + cf1 * (nf6[nv8[k]]-nf6[k]) + source6[k]; = nf7[k] - cf1 * (nf7[nv7[k]]-nf7[k]) + source7[k]; = nf8[k] - cf1 * (nf8[nv8[k]]-nf8[k]) + source8[k];</pre>

umag3.mec.utt.ro	Page 2
Printed by sofonea from flumag3.mec.utt.ro	wet9lf.c
	Jun 2 1999 20:19

Jun 2 1999 20:19	Page 1
/*************************************	

<pre>#include <stdio.h> #include <stdib.h> #include <math.h></math.h></stdib.h></stdio.h></pre>	
#include "wet9head.h"	
<pre>double compute_upwind_sources_bulk(int sigma, int index, int k, int nv, double nfil(), double nfil(),</pre>	
<pre>double compute_upwind_sources_boundary(int sigma, int index, int k, int nv,</pre>	
<pre>double lf_sources_bulk(int sigma, int index, int jnc, int jplus,</pre>	
double dummny, ux, uy, uu, neq, source; double fs1[9], fs2[9];	
fal(0) = tlfminus * nfl0[jminus] + tlfplus * nfl0[jplus]; fgl(1) = tlfminus * nfl1[jminus] + tlfplus * nfl1[jplus]; fgl(2) = tlfminus * nfl1[jminus] + tlfplus * nfl1[jplus]; fgl(3) = tlfminus * nfl1[jminus] + tlfplus * nfl1[jplus]; fgl(4) = tlfminus * nfl4[jminus] + tlfplus * nfl4[jplus]; fgl(6) = tlfminus * nfl5[jminus] + tlfplus * nfl6[jplus]; fgl(6) = tlfminus * nfl6[jminus] + tlfplus * nfl6[jplus]; fgl(7) = tlfminus * nfl8[jminus] + tlfplus * nfl1(jplus]; fgl(8) = tlfminus * nfl8[jminus] + tlfplus * nfl1(jplus);	
fs2[0] = tlfminus * nf20[jminus] + tlfplus * nf20[jplus]; fs2[1] = tlfminus * nf21[jminus] + tlfplus * nf21[jplus]; fs2[1] = tlfminus * nf22[jminus] + tlfplus * nf22[jplus]; fs2[3] = tlfminus * nf22[jminus] + tlfplus * nf23[jplus]; fs2[4] = tlfminus * nf24[jminus] + tlfplus * nf24[jplus]; fs2[5] = tlfminus * nf24[jminus] + tlfplus * nf25[jplus]; fs2[6] = tlfminus * nf25[jminus] + tlfplus * nf26[jplus]; fs2[7] = tlfminus * nf28[jminus] + tlfplus * nf26[jplus]; fs2[8] = tlfminus * nf28[jminus] + tlfplus * nf27[jplus];	
<pre>dummy = mass1 * (fs1[0]+fs1[1]+fs1[3]+fs1[4]+fs1[5]+</pre>	

_	4
5.	8
Ħ	ام
ပ္	_
ne l	
3.	
rinted by sofonea from flumag3.mec.utt.ro	
Ě	
2	
Ε	
<u>ē</u>	
4	
e	1
ō	
S	. 1
>	
5	
ě	
Æ	ret9lf.c
0	#
	2
	1

Jun 2 1999 20:19 Wet9lf.c Page 3	Jun 2 1999 20:19
<pre>uy = tminus2 * uyloc[jminus2] + tminus1 * uyloc[jminus1] + tout *uyloc[jloc]; uu = ux*ux + uy*uy; switch(sigma)</pre>	(1.000 + three over source = - cts
k]*uy) / cspee [[2]+fs1[3]+fa	(ecprodl[inc break; case 2; dummy = (ecx2[: neq = ww[index
.000 + three * hree_over_two ce = - ctaul ecprodl[index] k;	(1.000 + thru three_over_/ source = ctan
ex] * uy) / c [2]+fs2[3]+fs	Dreak;) return source;)
<pre>(1.000 + three * dummy + nine_over_two * dummy * dummy - three_over_two * uu / cspeed12); source = - ctau2 * (fs2[index] - neq) +</pre>	void lf_sources (doub) doub,
the state of the s	(int k;
<pre>double IX_SOURCES_IN(INC SIGNA, INC INCEX, int jloc, int jlus1, int jplus2, double tin, int oblust, double tplus2,</pre>	for (k=0; k <nnodes_{< td=""></nnodes_{<>
double nf10[], double nf11[], double nf12[], double nf15[], double nf14[], double nf15[], double nf16[], double nf21[], double nf18[], double nf20[], double nf22[], double nf23[], double nf23[], double nf28[], double nf28[], double nf28[],	
double dummy, ux, uy, uu, neq, source; double fs1[9], fs2[9];	
= tin * nf10[jloc] + tplus1 * nf10[jplus1] + tplus2 * nf1[= tin * nf11[jloc] + tplus1 * nf11[jplus1] + tplus2 * nf11 = tin * nf12[jloc] + tplus1 * nf12[jplus1] + tplus2 * nf12	case 0: sf11[k] = source11[k
niis [piusi] + tpius nf14[piusi] + tpius nf15[piusi] + tpius nf16[piusi] + tpius nf17[piusi] + tpius nf18[piusi] + tpius	sf21[k] = source21[k
+ tplus1 + t	sf12[k] = sourcel2[k
= tin * nr25[1]cc] + tplus1 * nr25[plus1] + tplus2 * = tin * nr26[1]cc] + tplus1 * nr26[plus1] + tplus2 * = tin * nr27[pluc1] + tplus1 * nr27[plus1] + tplus2 * = tin * nr27[ploc] + tplus1 * nr27[plus1] + tplus2 * + tin * nr28[juc2] + tplus1 * nr28[jplus1] + tplus2 * + tplus2 * nr28[jplus1] + tplus2[jplus1] + t	sf22[k] = source22[k
<pre>ux = tin * uxloc(jloc) + tplus1 * uxloc(jplus1) + tplus2 * uxloc(jplus2); uy = tin * uyloc(jloc) + tplus1 * uyloc(jplus1) + tplus2 * uyloc(jplus2);</pre>	R F13 [k]
uu = ux*ux + uy*uy; svitch(sigma)	m,
<pre>case 1: dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12; neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+</pre>	sf23[k] = source23[k

s1[7] ummy
<pre>three_over_two * uu / cspeedii/); source = - ctaul * (fsl[index] - neq) +</pre>
<pre>case 2: dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22; dummy = ww[index] * (fs2[0]+fs2[1]+fs2[3]+fs2[3]+fs2[5]+fs2[6]+ neq = ww[index] * fs2[7]+fs2[8], * ux + ecy2[index] * ux</pre>
<pre>(1.000 + three * dummy + hills_over_two * dummy * dummy - three_over_two * uu / cspeed22); source = - ctau2 * (fs2[index] - neq) +</pre>
return source;
<pre>void lf_sources(double nf10[], double nf11[], double nf12[], double nf13[], double nf14[], double nf15[], double nf16[], double nf17[], double nf18[], double nf20[], double nf21[], double nf22[], double nf23[], double nf24[], double nf25[], double nf26[], double nf27[], double nf28[])</pre>
int k;
<pre>for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>
<pre>sf10[k] = nf10[k]; sf20[k] = nf20[k]; sourcell[k] = - ctau1 * (nf10[k] - neq10[k]) -</pre>
<pre>switch(boundary_mode[k])</pre>
<pre>case 0: sfl1[k] = tlfminus1 * nfl1[nv3[k]] + tlfplus1 * nfl1[nv1[k]]; sourcel1[k] = lf_source_blk(l, 1, nv3[k], k, nv1[k], sourcel1[k] = lf_source_blk(l, 1, nv3[k], k, nv1[k],</pre>
Lightest, hill, nfl2, nfl4, nfl5, nfl6, nfl7, nfl8, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl1, nfl2, nf
TITIONUSS, TIPPLUSS, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
<pre>sf12[k] = tlfminus1 * nf12[nv4[k]] + tlfplus1 * nf12[nv2[k]]; source12[k] = lf_sources_bulk(1, 2, nv4[k], k, nv2[k],</pre>
LIMINUSI, CIPPIUSI, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf23, nf24, nf25, nf27, nf27, nf28); sf22[k] = tlfminus2 * nf22[nv4[k]] + tlfpius2 * nf22[nv2[k]]; source22[k] = lf_source2[lank[k], k, nv2[k],
Liminuss, Cippluss, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
<pre>sf13[k] = tlfminus1 * nf13[nv1[k]] + tlfplus1 * nf13[nv3[k]]; source13[k] = lf_sources_bulk[l, 3, nv1[k], k, nv3[k], +1fminus1, +1fminus1.</pre>
nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf1, nf20, nf21, nf22, nf23, nf24, nf25, nf27, nf28); sf23[k] = tlfminus2 * nf23[nv1[k]] + tlfplus2 * nf23[nv3[k]]; source23[k] = lf_sources_bulk(2, 3, nv1[k], k, nv3[k],

Jun 2 1999 20:19 wet9lf.c	Page 5
tifminus2, tifplus2, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
<pre>sf14[k] = tlfminus1 * nf14[nv2[k]] + tlfplus1 * nf14[nv4[k]]; source14[k] = lf_sources_Duk(1, 4, nv2[k], k, nv4[k], tlfminus1, tlfplus1, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf26, nf26, nf27, nf28); sf24[k] = tlfminus2 * nf24[nv2[k]] + tlfplus2 * nf24[nv4[k]]; source24[k] = lf_sources_Dulk(2, 4, nv2[k], k, nv4[k], nf10, nf11, nf12, nf18, nf16, nf17, nf18,</pre>	
<pre>sfl6[k] = tlfminus1 * nfl6[nv8[k]] + tlfplus1 * nfl6[nv6[k]]; source16[k] = lf_sources_bulk(1, 6, nv8[k], k, nv6[k],</pre>	
<pre>sf17[k] = tlfminus1 * nf17[nv5[k]] + tlfplus1 * nf17[nv7[k]]; source17[k] = lf_sources_bulk(1, 7, nv5[k], k, nv7[k], tlfminus1, tlfplus1, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf10, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); sf27[k] = tlfminus2 * nf27[nv5[k]] + tlfplus2 * nf27[nv7[k]]; source27[k] = lf_sources_bulk(2, 7, nv5[k], k, nv7[k], tlfminus2, tlfplus2, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);</pre>	
] = tlf 18[k] = 3] = tlf 28[k] =	
<pre>break; case 1: /* bottom wall */ sfil(k) = tifminus1 * nfil([nv3[k]] + tlfplus1 * nfil[[nv1[k]]]; sfil(k) = lf_sources_bulk(l, l, nv3[k], k, nv1[k],</pre>	

Jun 2 1999	9 20:19 wet9lf.c	Page 6
	n£20, n£21, n£22, n£23, n£24, n£25, n£26, n£27, n£28);	
	(k) - cfll * (nfl2[nv2[k]] - nfl2[k compute_upwind_sources_boundary(1, uxwall_bot, uywall_bot, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nf20, nf21, nf22, nf23, nf24, nf25, lk - cfl2 * (nf22[nv2[k]] - nf22[k]	
	compute_Upymil_bot, nfill, nfil, nfil2, nfil3, nfil4, nfil5, nf nf20, nf21, nf22, nf23, nf24, nf25, nf	
	<pre>sf13[k] = tlfminus1 * nf13[nv1[k]] + tlfplus1 * nf13[nv3[k]]; source13[k] = lf_sources_bulk[l, 3, nv1[k], k, nv3[k], *1_enum_tlfplus1</pre>	
	Lindings, Lippus, hill, nils, nils, nils, nils, nils, nilo, nill, nils, nils, nilo, nilo, nilo, nils, nilo,	
	tlfminus, tlfplust, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
	<pre>sfl4[k] = nfl4[k] - cfl1 * (nfl4[k] - nfl4[nv2[k]]); sourcel4[k] = compute_upwind_sources_bulk(1, 4, k, nv2[k],</pre>	
	sf24[k] = nf24[k] - cf12 * (nf24[k] - nf24[nv2[k]]); source24[k] = compute_upwind_sources_bulk(2, 4, k, nv2[k]); nf20, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
	s, k, nv5[k],	
	<pre>nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf11', nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf29); source25[k] = nf25[k] - cf12 * (nf25[nv5[k]) - nf25[k]); source25[k] = compute_upwind_sources_boundary(2, 5, k, nv5[k])</pre>	
	uxwall_bot, uywall_bot, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf27, nf28);	
****	<pre>sfl6[k] = nfl6[k] - cfl1 * (nfl6[nv6[k]] - nfl6[k]); source16[k] = compute withind sources_boundary(1, 6, k, nv6[k],</pre>	
	<pre>Lawari_Doc, uyeari_Doc, uyeari_Doc, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28), sf26(k) = nf26(k) - cf12 * (nf26(hv6(k)) - nf26(k)); source26(k) = compute upwind sources_boundary(2, 6, k, nv6(k),</pre>	••
	<pre>sel7(k) = nfl7(k) - cfll * (nfl7(k) - nfl7(nv5(k)); sourcel7(k) = compute_upwind_sources_bulk(1, 7, k, nv5(k),</pre>	
	sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]); source27[k] = compute_upwind_sources_bulk(2, 7, k, nv5[k], nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28)	
	<pre>sci8(k) = nf18(k) - cf11 * (nf18(k) - nf18[nv6(k]]); source18(k) = compute_upwind_sources_bulk(1, 8, k, nv6(k),</pre>	••

Printed by sofonea from flumag3.mec.utt.ro

Page 8

Pa	
wet9lf.c	Fig2 Fig2 Fig2, Fig2 Fig4 Fig5 Fig2 Fig3 Fig2 Fig2 Fig2 Fig2 Fig3 Fig3
Jun 2 1999 20:19	sf27[k] = nf source27[k] sf18[k] = nf source18[k] break; case 3: sf11[k] = nf source21[k] sf12[k] = nf source21[k] sf12[k] = nf source2[k] sf22[k] = nf source22[k] sf22[k] = nf source23[k] sf22[k] = nf source23[k] sf23[k] = nf source23[k] sf23[k] = nf source24[k] sf25[k] = nf source24[k] sf25[k] = nf source25[k] sf24[k] = nf source25[k]
Page 7	1
1999 20:19 Wet9If.c	source28[K] = compute_upwind_sources_balk[2, 8, K, nv6[k],

wet9lf.c

Jun 2 1999 20:19	wet9lf.c	Page 9
nf10, nf11, n nf20, nf21, n	nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
sf16[k] = nf16[k] - cf11 * source16[k] = compute_upwinc nf1, nf1, nf1, nf1, nf21, nf	<pre>i(k] - cfil * (nfi6(k) - nfi6(nv8(k))); compute_upwind_sources_bulk(1, 6, k, nv8(k), ifi0, nfil, nfi2, nfi2, nfi3, nfi4, nfi5, nfi6, nfi7, nfi8, nf20, nfi21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); i(k) - cfi2 * (nf26(k) - nf26[nv8(k)]); in cfi2 * (nf26(k) - nf26(nv8(k))); in cfi2 * (nf26(k) - nf26(k)); in cfi2 * (nf26(k)</pre>	
U	ompute_upwinc_sources_sour(k, o, k, nvelk), nfl0, nfl1, nfl2, nfl3, nfl4, nfl6, nfl6, nfl1, nfl8, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
sf17(k) = nf17(k) - cf11 * source17(k) = compute_upwin nf11, nn nf20, nf21, nn nf20, nf21, nn	(nf17[k] - nf17[nv5[k]]); d.sources_bulk(1, 7, k, nv5[k], f[l2, nf13, nf14, nf15, nf16, nf17, nf18, f22, nf23, nf24, nf25, nf26, nf27, nf28);	
sf27[k] = nf27[k] - cf12 * source27[k] = compute_upwin nf10, nf11, n nf20, nf21, n	nf18, nf28)	
sf18[k] = nf18[k] - cf11 * source18[k] = compute_upwin uwan1_left, i	<pre> k - cfil * (nfl8[nv8[k]] - nfl8[k]); compute_upwind_sources_boundary(1, 8, k, nv8[k],</pre>	
sf28[k] = nf28[k] - cf12 * source28[k] = compute_upwin nf10, nf11, n nf11, nf1	nico, mil, mil, mil, mil, milo, milo	
break; nf21, n.	f22, nf23, nf24, nf25, nf26, nf27, nf28);	
11[k] = urce13[k	nfil[k] - cfl1 * (nfil[k] - nfil[nv3[k]]); :] = compute_upwind_sources_bulk(i, 1, k, nv3[k], nfilo, nfil, nfi2, nfi3, nfi4, nfi5, nfi6, nfi7, nf18,	
<pre>sf21(k] = nf21(k] - cf12 * source21(k] = compute_upwin</pre>		
sf12[k] = tlfminus1 * nf12 source12[k] = 1f_sources_bu	* nfl2[nv4[k]] + tlfplus] * nfl2[nv2[k]]; ces_bulk(1, 2, nv4[k], k, nv2[k],	
LIMINUS, CI DÉZO, MÉZI, N SEZZ[k] = tléminus2 * nf22 Source22[k] = lf_Sources_Do	Liminus, Lipius, 1, 112, nf13, nf14, nf15, nf16, nf17, nf18, nf10, nf21, nf22, nf33, nf24, nf25, nf26, nf27, nf28); tfminus2 * nf22[nv4[k]]; + tfipus2 * nf22[nv2[k]]; = If_sources_Dulk[2, 2, nv4[k], k, nv2[k],	
tlíminus2, tl níl0, níl1, n ní20, ní21, n	nffinnus2, tifplus2, tifinnus2, tifplus2, nf10, nf11, nf12, nf13, nf14, nf15, nf26, nf27, nf28); nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
sf13[k] = nf13[k] - cf11 * source13[k] = compute_upwin	<pre>(nf13[nv3[k]] - nf13[k]); d_sources_boundary(1, 3, k, nv3[k], nvval</pre>	
### ##################################	nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, nfl0, nfl1, nfl2, nfl3, nfl9, nfs0, nfs2, nfs2, nfs6, nfs7, nfs8; nfl8, nfl1,	
UXWA11_1ert, nf10, nf11, n nf20, nf21, n	. uywari_letc, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
sfl4[k] = tlfminusl * nfl4 source14[k] = lf_sources_bu	* nfl4[nv2[k]] + tlfplus1 * nfl4[nv4[k]]; cs_bulk(1, 4, nv2[k], k, nv4[k],	
nf10, nf11, n		

Jun 2 1999 20	20:19 wet9lf.c Pag	Page 10
sf24[k]	######################################	
sf15 sour sf25 sour	<pre>sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]); source15[k] = compute_upwind_sources_bulk[1, 5, k, nv7[k],</pre>	
sf16[K] sourcel sf26[K]	= nf16[k] - cf11 * (nf16[nv6[k]] - 6[k] = compute_upwind_sources_bound uvaalleft, uvaalleft, nf10, nf11, nf12, nf13, nf14 = nf20, nf21, nf22, nf23, nf24 = nf26[k] - cf12 * (nf26[nv6[k]] 6[k] = compute_upwind_sources_bound uvaalleft, uvvallleft, nf10, nf11, nf12, nf13, nf14 nf20, nf21, nf22, nf23, nf24	
sfl7 sour	<pre>sel7[k] = nf17[k] - cf11 * (nf17[nv7[k]] - nf17[k]); source17[k] = compute_upwind_sources_boundary(1, 7, k, nv7[k],</pre>	
~	sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]); source18[k] = compute_upwind_sources_bulk(1, 8, k, nv6[k], nf10, nf11, nf12, nf23, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28]; sf28[k] = nf28[k] - cf12 * (nf28[k] - nf26[k]); source28[k] = compute_upwind_sources_bulk(2, 8, k, nv6[k], nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf18, break;	
yoid lf(double double	sf0[], double sf1[], double sf2[], sf3[], double sf4[], double sf8[], sf6[], double sf7[], double sf8[], source3[], double source4[], double source5[], source6[], double source4[], double source5[], nff0[], double nff1[], double nff2[], nff3[], double nff1[], double nff5[], nff8[], double nff7[], double nff8[])	
int k; for (k=0; { nff0 nff1 nff3 nff3 nff3	<pre>kcnnodes_all; k++) kl = sf0[k] + source0[k]; kl = sf1[k] + source1[k]; kl = sf2[k] + source2[k]; kl = sf2[k] + source2[k]; kl = sf4[k] + source3[k]; kl = sf4[k] + source4[k];</pre>	

Printed by sofonea from flumag3.mec.utt.ro

wet9iw.c

2
£
i i
_
E E
~
හි
Ø
트
≢
=
ō
₽
ಹ
٣
\$
8
>
Ω
줐
¥
₹
Δ.

wet9lw.c

May 31 1999 19:07 Wet9lw.c Page 1	May 31 1999 19:07
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	fa2[0] = tminus +
* wet91w.c	11 11
/ 程存以在根据的有效的现在分词有限的现在分词的 医克朗特氏性 医克朗特氏性 医克朗特氏性 医克朗特氏性 医克朗特氏性 医克朗特氏性 医克朗特氏性 医克朗特氏病 人名英格兰斯特特 医克朗特氏征 医克朗特氏征检查氏征检查检查检查检查检查检查检查检查检查检查检查检查检查检查检查检查检查	fs2[3] = tminus * fs2[4] = tminus *
<pre>#include <stdio.h> #include <stdlib.h> #include <math.h></math.h></stdlib.h></stdio.h></pre>	
#include "wet9head.h"	ISZ[8] = EMINUS * I
<pre>double compute_upwind_sources_bulk(int sigma, int index, int k, int nv,</pre>	Case 0: dummy = mass1 * mass2 * (fs2 mass2 * (fs2 ux = (mass1 *
<pre>double compute_upwind_sources_boundary(int sigma, int index, int k, int nv,</pre>	mass2 * uy = (mass1 * mass2 *
<pre>double compute_upwind_sources_boundary_in(int sigma, int index, int k, int nv,</pre>	
<pre>double compute_upwind_sources_boundary_out(int sigma, int index, int nv, int k,</pre>	<pre>uy = uywail_to</pre>
<pre>double lw_sources_bulk(int sigma, int index, int jminus, int jloc, int jplus, double tminus, double tloc, double tplus, double nfl0[], double nfl1[], double nfl2[], double nfl3[], double nfl4[], double nfl5[], double nfl2[], double nfl7[], double nfl8[], double nfl2[], double nfl2[], double nfl2[], double nfl2[], double nfl2[], double nfl2[], double nfl2[], double nfl2[], double nfl2[], double nfl2[],</pre>	(1.000 + thr (1.000 + thr three_over_ source = - ct (ecprod1[in break; case 2: dummy = (ecx2[neq = ww[index
<pre>{ double dummy, ux, uy, uu, neq, source; double fs1[9], fs2[9];</pre>	+ 6
<pre>fs1[0] = tminus * nf10[jminus] + tloc * nf10[jloc] + tplus * nf10[jplus]; fs1[1] = tminus * nf11[jminus] + tloc * nf11[jloc] + tplus * nf11[jplus]; fs1[2] = tminus * nf12[jminus] + tloc * nf12[jloc] + tplus * nf12[jplus]; fs1[3] = tminus * nf13[jminus] + tloc * nf13[jloc] + tplus * nf13[jplus]; fs1[4] = tminus * nf14[jminus] + tloc * nf14[jloc] + tplus * nf14[jplus]; fs1[5] = tminus * nf14[jminus] + tloc * nf14[jloc] + tplus * nf14[jplus]; fs1[6] = tminus * nf16[jminus] + tloc * nf16[jloc] + tplus * nf16[jplus];</pre>	source = - coa break; break; return source; }
= tminus * nfl7[jminus] + tloc * nfl7[jloc] + tplus * tminus * nfl8[jminus] + tloc * nfl8[jloc] + tplus *	double lw_sources_ou

fs2[0] = tminus * nf20[jminus] + tloc * nf20[jloc] + tplus * nf20[jplus]; fs2[1] = tminus * nf21[jminus] + tloc * nf21[jloc] + tplus * nf21[jplus]; fs2[2] = tminus * nf22[jminus] + tloc * nf22[jloc] + tplus * nf22[jplus]; fs2[3] = tminus * nf23[jminus] + tloc * nf23[jloc] + tplus * nf23[jplus]; fs2[4] = tminus * nf24[jminus] + tloc * nf24[jloc] + tplus * nf24[jplus]; fs2[6] = tminus * nf25[jminus] + tloc * nf25[jloc] + tplus * nf25[jplus]; fs2[6] = tminus * nf27[jminus] + tloc * nf25[jloc] + tplus * nf26[jplus]; fs2[7] = tminus * nf27[jminus] + tloc * nf28[jloc] + tplus * nf28[jplus]; fs2[8] = tminus * nf28[jminus] + tloc * nf28[jloc] + tplus * nf28[jplus];
lary.
<pre>mass2 * (182(0)+182(1)+182(1)+182(1)+182(1)+</pre>
<pre>ts2[7]*ecx2[7]+fs2[8]*ecx2[8])/tau2) / dummy; uy = (mass1 * (fsx[1]*ecy1[1]*fs1[2]*ecy1[2]+fs1[3]*ecy1[3]+ fs1[4]*ecy1[4]+fs1[8]*ecy1[5]+fs1[6]*ecy1[6]+ fs1[7]*ecy1[7]+fs1[8]*ecy1[8])/tau1 + mass2 * (fs2[1]*ecy2[1]+fs2[2]*ecy2[2]+fs2[3]+ fs2[1]*ecy2[1]+fs2[8]*ecy2[5]+fs2[3]+ fs2[7]*ecy2[7]+fs2[8]*ecy2[6]+fs2[6]*ecy2[6]+ fs2[7]*ecy2[7]+fs2[8]*ecy2[8]</pre>
11 11 60
<pre>k*ux + uy*uy; (sigma)</pre>
+++
tree_over_two * uu / cspeedil; ce = - ctaul * (fsl[index] - neq) + ecprodi[index] - csforce_x * ux - csforce_y * u
mmy = (ecx2[index] * ux y = ww[index] * (fs2[0] fs2[7] (1.000 + three * dummy
<pre>tnree_over_two * u / ospeed22); source = - ctau2 * (fs2[index] - neq) +</pre>
return source;
<pre>double lw_sources_out(int sigma, int index, int jminus2, int jminus1, int jloc,</pre>

wet9lw.c

May 31 1999 19:07

က္မ			
May 31 1999 19:07 Wet9Iw.c	<pre>double tminus2, double tminus1, double tout, double nf10[], double nf12[], double nf13[], double nf15[], double nf15[], double nf15[], double nf15[], double nf15[], double nf20[], double nf21[], double nf22[], double nf20[], double nf21[], double nf22[], double nf20[], double nf21[], double nf25[], double dummy, ux, uy, uu, neq, source; double fs1[9], fs2[9];</pre>	tminus2	<pre>ux = tminus2 * uxloc[jminus2] + tminus1 * uxloc[jminus1] + tout *uxloc[jninus2], uy = tminus2 * uyloc[jminus2] + tminus1 * uyloc[jminus1] + tout *uyloc[jloc]; uu = ux*ux + uy*uy; switch(sigma) case 1: dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12; neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+fs1[6]+</pre>

	e fs1[9]. fs2[9]	10	= tin * uxloc[jloc] + tplus1 * uxloc[jplus1] + tplus2 * uxloc[: = tin * uyloc[jloc] + tplus1 * uyloc[jplus1] + tplus2 * uyloc[: = ux*ux + uy*uy; cch(sigma)	1: my = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12; my = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs fs1[7]+fs1[8] *	<pre>double nf10[], double nf11[], double nf12[], double nf13[], double nf14[], double nf15[], double nf16[], double nf17[], double nf18[], double nf23[], double nf21[], double nf22[], double nf23[], double nf21[], double nf25[], double nf26[], double nf27[], double nf28[]) nf10[k]; nf20[k]; nf20[k]; nf20[k]; nf20[k]; k] = - ctaul * (nf10[k] - neq10[k]) - neq10[k] k] = - ctaul * (nf20[k] - neq20[k]) - neq10[k] k] = - ctaul * (nf20[k] - neq20[k]) + neq20[k]</pre>
--	------------------	----	---	--	--

wet9lw.c

May 31	1 1999 19:07 wet9lw.c	Page 5
	<pre>sfil(k) = tlminus * nfil(nv3[k)] + tlcenter * nfil(k) + sfil(k) = tlminus * nfil(nv1[k)] +</pre>	
	sourcell[k] = 1w_sources_bulk[1, 1, nv3[k], k, nv1[k], + 1 w 1 1 1 1 1 1 1 1 1	
	nf10, nf21, nf23, nf14, nf15, nf16, nf17, nf20, nf27.	, nf18,
	sf21[k] = t2minus * nf21[nv3[k]] + t2center * nf21[k] + +2minus * nf21[v][k]];	
	<pre>source21(k) = 1w_sources_bulk(2, 1, nv3(k), k, nv1(k),</pre>	
	nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf20, nf21, nf22, nf23, nf24, nf25, nf27,	, nf18, , nf28);
	<pre>sf12[k] = tlminus * nf12[nv4[k]] + tlcenter * nf12[k] + +1alis * nf12[nv2[k]].</pre>	
	sourcel2[k] = lw_source_bulk[1, 2, nv4[k], k, nv2[k],	
	nflo,	, nf18,
	sf22[k] = t2minus * nf22[nv4[k]] + t2center * nf22[k] + t2plus * nf22[nv2[k]];	
	<pre>source22[k] = 1w_sources_bulk(2, 2, nv4[k], k, nv2[k],</pre>	
/	n£10, n£11, n£12, n£13, n£14, n£15, n£16, n£17, n£ n£20, n£21, n£22, n£23, n£24, n£25, n£26, n£27, n£	, nf18, , nf28);
	sf13[k] = tlminus * nf13[nv1[k]] + tlcenter * nf13[k] +	
	<pre>tlplus * nri3[nv3 K]]; source13[k] = lw_sources_bulk(1, 3, nv1[k], k, nv3[k],</pre>	
	timinus, tlcenter, tlplus, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf1	nf18,
	nf20, nf21, nf22, nf23, nf24, nf25, nf25, nf26, nf27, sf23[k] + t2center * nf23[k] +	, nf28);
	source23[k] = Iw sources bulk(2, 3, nv1[k], k, nv3[k],	
	raminus, recencer, rispius, nf15, nf16, nf17, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27,	, nf18,
	sfl4[k] = tlminus * nfl4[nv2[k]] + tlcenter * nfl4[k] +	
	source14[k] = lw_sources_bulk[1, 4, nv2[k], k, nv4[k],	
	Liminus, r.cencer. Lipius, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27,	, nf18,
	sf24[k] = t2minus * nf24[nv2[k]] + t2center * nf24[k] + t20lus * nf24[nv4[k]];	
	<pre>source24[k] = iw_sources_bulk(2, 4, nv2[k], k, nv4[k], t2minus, t2center, t2plus, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27,</pre>	', nf18,
	sf15[k] = tlminus * nf15[nv7[k]] + tlcenter * nf15[k] +	
	source15[k] = lw_source = 1 w_source = 1 w_s	
	Lininus, Licenest, Lipius, Lini, pill2, nil3, nil4, nil5, nil6, nil nil0, nil1, nil2, nil3, nil4, nil25, nil6, nil	/, nf18,
	sf25[k] = t2minus * nf25[nv7[k]] + t2center * nf25[k] + t2los * nf25[nv7[k]] + t2center * nf25[k] + t2los * nf25[nv7[k]] + t2center * nf2[k] + t2c	
	Sourcezoln) - m_sourcezolni, topius, topius, topius, nfilo, nfilo	/, nf18, /, nf28);
	<pre>sfl6[k] = tlminus * nfl6[nv8[k]] + tlcenter * nfl6[k] + tlplus * nfl6[nv6[k]];</pre>	

3.mec.utt.ro	Page 6
Printed by sofonea from flumag3	wet9lw.c
	May 31 1999 19:07

##############################
--

May 31 1999 19:07	sf28[k] = t source28[k]	/* sf11[k] = t sourcel1[k] sf21[k] = t	<pre>af12[k] = ni source12[k] sf22[k] = ni source22[k] sf13[k] = t</pre>	source13[k] sf23[k] = { source23[k] sf14[k] = n	sf24[k] = n; source24[k] sf15[k] = n; source15[k]	sf25[k] = n source25[k] sf16[k] = n source16[k]
Page 7						
wet9lw.c	nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); t2minus * nf23[nv1[k]] + t2center * nf23[k] + t2plus * nf23[nv3[k]];] = lw_sources_bulk(2, 3, nv1[k], k, nv3[k], t2minus, t2center, t2plus, nf18, nf16, nf17, nf18, nf10, nf21, nf22, nf33, nf14, nf25, nf26, nf21, nf28);	<pre>toutlminus2 * nf14[nv2[k]]] + toutlminus1 * nf14[nv2[k]] + toutlminus1 * nf14[k]] + toutlminus1 * nf14[k]] + loutlminus2 * nf14[k] + nf10, nf11, nf2, nf3, nf14, nf15, nf16, nf17, nf18, nf10, nf11, nf22, nf23, nf24, nf25, nf26, nf27, nf28); toutZminus2 * nf24[nv2[k]] + toutZminus1 * nf24[nv2[k]] + tout2 * nf24[nf25] + tout2 * nf24[n</pre>	tinl * nfi5[k + tinlplusl * nfi5[nv5[k]] + tinlplus2 * nfi5[nv5[hv5[k]];	tinl * nfi6[k] + tinlplusl * nfi6[nv6[k]] + tinlplusz * nfi6[nv6[k]];	toutim toutim = lw_ tou nfi toutim toutim	= lw sources out(2, 7, nv5[nv5[k]], nv5[k], k, tout2minus2, tout2minus1, tout2, nf13, nf14, nf15, nf16, nf17, nf18, nf10, nf11, nf22, nf23, nf24, nf25, nf26, nf27, nf28) tout1minus2 * nf18[nv6[k]] + tout1minus1 * nf18[nv6[k]] + tout1minus1 * nf18[k]; cout1 * nf18[k]; cout1 * nf18[k]; l] = lw.sources_out(1, 8, nv6[nv6[k]], nv6[k], k, tout1minus2, tout1minus1, tout1,
May 31 1999 19:07	sf23[k] = source23[k]	sf14[k] = source14[k] sf24[k] = source24[k]	sf15[k] = source15[k] sf25[k] = source25[k]	<pre>sf16[k] = source16[k] sf26[k] = source26[k]</pre>	sfl7[k] = source17[k] sf27[k] =	source27[k] sf18[k] = source18[k]

ובכי חוויו ח	Page
IIIago.II	
11011	
SOLUTION	
rillied by solding flori intrings.	wet9lw.c
	We

		1
		88
<pre>nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,</pre>	sourcel2[k] = nf12[k] - cf11 * (nf12[nv2[k]] - nf12[k]); sourcel2[k] = compute_unwind_sources_boundary(1, 2, k, nv2[k],	
nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); plus * nf23[nv1[k]] + t2center * nf23[k] + plus * nf23[nv1[k]] + t2center * nf23[k] + tplus * nf23[nv1[k]]; tzminus, t2center, t2plus, t2pl		wet9lw.c

wet9lw.c

May 31 1999 19:07 Wet9lw.c Page 9	May 31 1999 1
<pre>uxwall_bot, uywall_bot, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf23, nf23, nf24, nf26, nf27, nf28); sf26[k] = nf26[k] - cf12 * (nf26[kv6[k]] - nf26[k]); source26[k] = compute_upwind_sources_boundary(2, 6, k, nv6[k], uxwall_bot, uywall_bot, nt10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);</pre>	\$224 sour
<pre>sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]); source17[k] = compute_upwind_sources_bulk[1, 7, k, nv5[k],</pre>	sour sf25
= n 8[k]	 sf16 sour
<pre>sf28[k] = nt28[k] - cf128[k] - nt28[nv6(k]); source28[k] = compute_upwind_sources_bulk(2, 8, k, nv6[k],</pre>	 sf26 sour
<pre>/* sfll[k] = tlminus * nfll[nv3[k]] + tlcenter * nfll[k] + tlplus * nfll[nv1[k]]; sourcell[k] = lw_sources_bulk(i, i, nv3[k], k, nv1[k], tlminus, tlcenter, tlolus,</pre>	 sf17 sour
<pre>nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,</pre>	 sf27 sour
<pre>nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); sf12[k] = nf12[k] - cf11 * (nf12[nv2[k]] - nf12[k]); source12[k] = compute_upwind_sources_boundary_in(1, 2, k, nv2[k],</pre>	 sf18 sour
<pre>uxwall_bot, uywall_bot, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); sf22[k] = nf22[k] - cf12 * (nf22[nv2[k]] - nf22[k]); source22[k] = compute_upwind_sources_boundary_in(2, 2, k, nv2[k], uxwall_bot, uywall_bot, nf18, nf16, nf17, nf18, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18,</pre>	 sf28 sou
	 bre
tiplus milionality circuica. milionality tiplus a filiality (k) 3 [k] = lw.sources_bulk(1, 3, nv1[k], k, nv3[k], timinus, ticenter, tiplus,	 case case Sf1
nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf11, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); sf23[k] = t2minus * nf23[nv3[k]] + t2center * nf23[k] + t2plus * nf23[nv3[k]];	 inos
source23[k] = lw_sources_bulk(z, s, nvt[k], k, nv3[k],	 sf2.
<pre>sf14(k) = nf14(k) - cf11 * (nf14(k) - nf14(nv2(k))); source14(k) = compute_upwind_sources_boundary_out(1, 4, nv2(k), k,</pre>	 s£1.

May 31 1999 19:07	wet9lw.c Page 10
sf24[k] = nf24 source24[k] = 0	nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); 4[K] - cf12 * (nf24[K] - nf24[nv2[K]]); *compute_upwind_sources_boundary_out(2, 4, nv2[K], k, uxwall_bot, uywall_bot, nf10, nf11, nf22, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);
sf15[k] = nf15[k] = com source15[k] = com nf1 sf25[k] = nf25[k] source25[k] = com nra	occupate occupate
sf16[k] = nf16[k]	- cfll nute_upw ll_bot, nfll, nfll, cfll nute_upw ll_bot, nfll,
sourcel7[k] = nf17[k] sourcel7[k] = com ntm nf1 sf27[k] = nf27[k] source27[k] = com nxm	<pre>[k - cfll * (nf17[k] - nf17[nv5[k]]); compute_upwind_sources_boundary_out(1, 7, nv5[k], k, uxwall_bot, uywall_bot, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf11, nf21, nf23, nf24, nf25, nf26, nf27, nf28); (kl - cfl2 * (nf27[k] - nf27[nv5[k]]); compute_upwind_sources_boundary_out(2, 7, nv5[k], k, uxwall_bot, uywall_bot, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf11, nf12, nf33, nf24, nf25, nf26, nf27, nf28);</pre>
source18[k] = nf18[k] source18[k] = comg ntw ntw ntw source28[k] = nf28[k] source28[k] = comg nt11 nf11	<pre>f18[k] - cfil * (nfi8[k] - nfi8[nv6[k]]);</pre>
break; /* top sfl1[k] = tlmi sourcel1[k] = 1 sf21[k] = tlmi sf21[k] = t2mi source21[k] = t2mi sf12[k] = t2mi	top wall */ tlminus * nfil[nv3[k]] + tlcenter * nfil[k] + tlpius * nfil[nv1[k]] + tlcenter * nfil[k] + tlpius * nfil[nv1[k]], nv3[k], k, nv1[k], tlminus, tlcenter, tlplus, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil, tzminus * nfil(nv1[k]] + tzcenter * nfil(k) + tzplus * nfil(k)] + tzcenter * nfil(k) + tzplus * nfil(k)] + tzcenter * nfil(k) + tzplus * nfil(k)] +

Printed by sofonea from flumag3.mec.utt.ro

ag3.mec.utt.ro
a from flumag
by sofone
Printed

wet9lw.c

Page 12

May 31 1999 19:07 wet9lw.c	Page 11	-	May 31 1999 19:07
<pre>toutlminus1 * nfl2[nv4[k]] +</pre>	k]], nv4[k], k,		sf17[k] = source17[k
hriu, hriz,	nf25, nf26, nf27, nf28); +		s£27[k] =
<pre>tout2 * nf22[k] > vv4[nv4[k]], nv4[k], k, tout2minus2, tout2minus1, tout2,</pre>	t,], nv4[k], k, t,, nf1s, nf16, nf17, nf18, nf2s, nf26, nf27, nf28);		source27[k
sf13[k] = tlminus * nf13[nv1[k]] + tlcenter tlplus * nf13[nv3[k]]; source13[k] = lw sources bulk(l, 3, nv1[k],	r * nf13[k] + k, nv3[k],		sf18[k] = source18[k
tlminus, tlcenter, tlplus, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); sf23[k] = t2minus * nf23[nv3[k]] + t2center * nf23[k] + t2plus * nf23[nv3[k]];	nf15, nf16, nf17, nf18, nf25, nf26, nf27, nf28); r * nf23[k] +		sf28[k] =
source23[k] = lw_sources_bluk(k, 3, nvl[k], taminus, t.Zenter, t.Zplus, nfl0, nfl1, nfl2, nfl3, nfl4, nf20, nf21, nf22, nf24,	k, nv3(k], nf15, nf16, nf17, nf18, nf25, nf26, nf27, nf28);		sourcezsik
<pre>sfl4[k] = tin1 * nfl4[k] + tin1plus1 * nfl4[nv4[k]] +</pre>	.4[nv4[k]] + , nv4[nv4[k]],	.,-	/* sf11[k] =
tini, tiniplusi, tiniplusi, tiniplusi, nf13, nf13, nf14, nf20, nf21, nf22, nf23, nf24, nf21, nf21, nf21, nf24 lk] = tin2 * nf24(k] + tin2plusi * nf24	nfl5, nfl6, nfl7, nfl8, nf25, nf26, nf27, nf28); 4[nv4[k]] +		source11[k
tin2plus2 * nf24[nv4[ky]]; source24[k] = lw_source2, in(2, 4, k, nv4[k], nv4[nv4[k]], tin2, tin2plus1, tin2plus2, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	, nv4[nv4[k]], nf15, nf16, nf17, nf18, nf25, nf26, nf27, nf28);	*****	sf21[k] = source21[k
sf15[k] = toutlminus2 * nf15[nv7[nv7[k]]] + toutlminus1 * nf15[nv7[k]] + toutl * nf15[nv7[k]] + toutl* * nf15[nv7] k]	+ 		sf12[k] = source12[k]
sourcell(s) = IM_Sources_cut(t,), M(10)(15); M(10)(15)	rij, nejej, neje, neje, neje, neje, neje, neje, neje, neje); +		sf22[k] = source22[k]
<pre>tout2 * nf25[k]; source25[k] = lw.sources.out(2, 5, nv7 nv7 l) tout2minus2, tout2minus1, tout nf10, nf11, nf12, nf13, nf14, nf20, nf21, nf22, nf23, nf24,</pre>	[k]], nv7[k], k, ht2, nf15, nf16, nf17, nf18, nf25, nf26, nf27, nf28);	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	sfl3[k] = sourcel3[
<pre>sf16[k] = toutlminus2 * nf16[nv8[k]]] +</pre>	+ [k]], nv8[k], k, t1, , nf15, nf16, nf18,		sf23[k] = source23[l]
nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, tout2minus2 * nf26[nv8[k]] + tout2minus1 * nf26[nv8[k]] + tout2minus1 * nf26[k]] + source26[k] = lw_2ource2_out(2, 6, nv8[nv8[k]], nv8[k], k, tout2minus2, nf22, nf23, nf24, nf25, nf26, nf27, nf22, nf22, nf22, nf22, nf26, nf27,	27,		source14[k] =

<pre>sf17(k) = tinl * nf17(k) + tinlplus1 * nf17(nv7(k)] + tinlplus2 * nf7(nv7[nv7(k)]); source17(k) = iw_sources_inl, 7, k, nv7(k), nv7(nv7(k)], tinl, tinlplus1, tinlplus2, tinl, tinlplus1, tinlplus2, nf10, nf11, nf22, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf11, nf22, nf13, nf24, nf25, nf26, nf27, nf28); sf27(k) = tin2 * nf27(k) + tin2plus1 * nf27(nv7(k)] + tin2plus2 * nf27(nv7(nv7(k)]); source27(k) = lw_sources_in(2, 7, k, nv7(k)) tin2, tin2plus1, tin2plus2, nf10, nf12, nf12, nf14, nf15, nf16, nf17, nf18, nf10, nf11, nf22, nf13, nf24, nf25, nf26, nf27, nf28);</pre>	<pre>sfl8[k] = tinl * nf18[k] + tinlplus1 * nf18[nv8[k]] +</pre>	/* sfil(k) = timinus * nfil(nv3(k)) + tloenter * nfil(k) + sfil(k) = tiplus * nfil(mv1(k)); sourcell(k) = lw_sources_bulk(l, 1), nv3(k), k, nv1(k), timinus, tloenter, tiplus, nfil(n, nfil, nfil, nfil, nfil, nfil, nfil, nfil, nfil(n, nfil, nfil, nfil, nfil, nfil, nfil, nfil, scurcel(k) = tzminus * nfil(mv3(k)) + tzcenter * nfil(k) + tzplus * nfil(mv3(k)) + tzcenter * nfil(k) + tzplus * nfil(mv1(k)); sourcel(k) = lw_sources_bulk(l, 1, nv3(k), k, nv1(k), tzminus, tzcenter, tzplus, nfil(n, nfil, nfil, nfil, nfil, nfil), nfils, nfils	<pre>sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]); source12[k] = compute_upwind_sources_bulk(1, 2, k, nv4[k],</pre>	sfl3[k] = tlminus * nfl3[nv1[k]] + tlcenter * nfl3[k] + tlplus * nfl3[nv3[k]; sources]3[k] = lw_sources_bulk[i, 3, nv1[k], k, nv3[k], tlminus, tlcenter, tlplus, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, t2plus * nfl2]iv1[k] + t2center * nfl2[k] + t2plus * nfl2]iv1[k]] + t2center * nfl2[k] + t2plus * nfl2[nv3[k]]; source23[k] = lw_sources_bulk[2, 3, nv1[k], k, nv3[k], t2minus, t2center, t2plus, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, nfl0, nfl1, nfl2, nfl2, nfl2, nfl2, nfl2, nfl2, nfl2,	<pre>sfl4[k] = nfl4[k] - cfl1 * (nfl4[nv4[k]] - nfl4[k]); sourcel4[k] = compute_upwind_sources_boundary(1, 4, k, nv4[k],</pre>
---	--	--	--	--	--

wet9lw.c

40

Page 14

May 31 1999 19:07 wet9lw.c	### ##################################	111111111111111111111111111111111111111
May 31 1999 19:07 Wet9IW.C Page 13	succest(N) = nacate, comparisones, promoter, control of the nacate of magning and compared to the nacate of magning and compared to the nacate of magning and compared to the nacate of the nacet of the nacate of the nacet of the nace	uxwall_top, uywall_top,

y 31 1999 19:07 Wet9lw.c	7,
nf15, nf16, nf17, nf25, nf26, nf27,	~ ~
i(k) + k], f16, nf17,	n f.1
nf25, nf26, nf27, r * nf23[k] + k, nv3[k],	7, n£28);
t2minus, t2center, t2plus, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27,	~ ~
ry_in(l, 4, k, nv4[_
f16, nf17, f26, nf27, , , k, nv4[7, nf18, ?7, nf28); v4[k],
uxwall_cop, uyaall_cop, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf	17, nf18, 17, nf28);
<pre>sfi5[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]); source15[k] = compute, upwind, sources_boundary_out(1, 5, nv7[k],</pre>	
nf14, nf15, nf16, nf17, nf24, nf25, nf26, nf27, nf25[nv7[k]]); oundary_out(2, 5, nv7[k]	
uxwall_top, uywall_top, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27,	7.
<pre>sf16[k] = nf16[k] - cfll * (nf16[k] - nf16[nv8[k]]); source16[k] = compute_upwind_sources_boundary_out(1, 6, nv8[k],</pre>	[X]
nf14, nf15, nf16, nf17, nf24, nf25, nf26, nf27, nf26[nv8[k]]); oundary_out(2, 6, nv8[k]	
uxwall_top, uywall_top, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf20, nf21, nf22, nf23, nf24, nf26, nf27,	7, nf 7, nf
- nf17[k]); dary_in(1, 7, k, nv7[17
fil, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, f21, nf22, nf23, nf24, nf25, nf26, nf27, f12 * (nf27[nv7[k]] - nf27[k]); nowind sources boundary in (2, 7, k, nv7[r- 5
uxwall_top, uywall_top, nfl0, nfl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27,	77

4

wet9lw.c
nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); = tin1 * nf11[k] + tin1plus1 * nf11[nv1[k]] +
<pre>tinlplus2 * nfil(nvl(nvl(k)); sourcell[k] = lw_sources_int, 1, k, nvl(k), nvl(nvl(k)),</pre>
<pre>source21(k) = lw.sources.in(2, 1, k, nv1[k), nv1[nv1[k]], tin2, tin2plus1, tin2plus2, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);</pre>
<pre>sfl2[k] = tlminus * nf12[nv4[k]] + tlcenter * nf12[k] +</pre>
tzplus * nrzzlnv2(k]; [k] = lw_sources_bulk(2, z.nv4[k], k, nv2[k],
<pre>sf13[k] = toutiminus2 * nf13[nv1[kv]] +</pre>
<pre>toutlminus2, toutlminus1, tout1, nf10, nf11, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf23, nf24, nf25, nf26, nf27, nf28); tout2minus2 * nf23[nv1[nv1[k]]] + tout2minus1 * nf23[nv1[k]] +</pre>
<pre>tourz * ni23 k ; source23[k] = lw_sources_ources. tout2minus2, tout2minus1, tout2,</pre>
<pre>sf14[k] = tlminus * nf14[nv2[k]] + tlcenter * nf14[k] + tlplus * nf14[nv4[k]]; source14[k] = lw_sources_bulk(1, 4, nv2[k], k, nv4[k], tlminus, tlcenter, tlplus, tlentus, tlcenter, tlplus, tlentus, tlcenter, tlplus,</pre>
##################################
<pre>tin1 * nf15[k] + tiniplus1 * nf15[nv5[k]] + tiniplus2 * nf15[nv5[nv5[k]]]; [k] = lw_sources_in(l, 5, k, nv5[k], nv5[k]), tin1, tin1plus1, tin1plus2,</pre>
<pre>nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); tin2plus * nf25[r] + tin2plus1 * nf25[r] + tin2plus2 * nf25[r] + tin2plus1 * nf25[r] + source25[k] = lm_sources_ln[2, 5, k, nv5[k]]; tin2, tin2plus1, tin2plus2,</pre>

May 31 1999 19:07	wet9lw.c	Page 16
	nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf24, nf25, nf26, nf27, nf28);	
sf16[k] =	<pre>toutiminus2 * nfis[nv8[nv8[k]]] + toutiminus1 * nfis[nv8[k]] +</pre>	
source16[}	<pre>[k] = Iw_sources_out(1, 6, nv8[nv8[k]), nv8[k], k,</pre>	
sf26[k] =		
source26[k]	<pre>Lout2 * nize(k!); classing for the profile in the profile in</pre>	
sf17[k] =	<pre>toutlminus2 * nf17[nv5[nv5[k]]] + toutlminus1 * nf17[nv5[k]] +</pre>	
source17[k] sf27[k] =		
source27[k]		
sf18[k] =	<pre>= tin1 * nf18[k] + tin1plus1 * nf18[nv8[k]] + tin1plus2 * nf18[nv8[k]]]; ti = 1 * * * * * * * * * * * * * * * * * *</pre>	
	tini, tiniplusi, tiniplus2, nf10, nf11, nf12, nf13, nf14, nf15, nf nf20, nf21, nf22, nf23, nf24, nf25, nf	
sf28[k] = source28[<pre>= tin2 * nf28[k] + tin2plus1 * nf28[nv8[k]] + tin2plus2 * nf28[nv8[kv]]; [k] = lw sources in(2, 8, k, nv8[k], nv8[nv8[k]],</pre>	
	tin2, tin2plus1, tin2plus2, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf20, nf21, nf22, nf23, nf24, nf25, nf26,	
break; case 4: sfl1[k] =		
-	<pre>toutlminus1 * nf11[nv3[k]] +</pre>	
sourcell[k]	<pre>= lw_sources_out(1, l, nv3[nv3[k]], nv3[k], k, toutlainus2_ toutlainus1, toutl, nf10, nf11, nf12, nf13, nf14, nf15, nf16</pre>	
sf21[k] =	nrIO, nrIO, nrIO, nrIO, nrIO, nrIO, nrIO, nrIO, nrIO); toutZminus1 * nfII[nv3]kr]] + toutZminus1 * nfII[nv3]kr] +	
source21[k]	k] = lw_souces_out(2, 1, nv3[nv3[k]], nv3[k], k, toutZminns2, toutZminus1, tout2, nf10, mf11, nf22, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	
•	t1n	
source12[<pre>[k] = lw_sources_bulk(1, 2, nv4[k], k, nv2[k], tlminus, tlcenter, tlplus, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf10, nf12, nf12, nf13, nf14, nf15, nf16, nf17, nf18,</pre>	
sf22[k] =	nizu, nizi, nizz, nizs, nizs, nizs, nizs, tizi, t2minus * nf22[nv4[k]] + t2center * nf22[k] +	

Printed by sofonea from flumag3.mec.utt.ro

42 wet9lw.c

wet9lw.c

1.3	
c.ut	
3.mec	
띥	
nac	
ᆵ	
Ε	
\$	
ea	
<u>6</u>	
Sot	
ò	
b	
Ĕ	
ሷ	

wet9lw.c

May 31 1999 19:07	s£18[k] =	source18[k	sf28[k] =		break;	void lw(double sf0[]	double sourc double sourc double sourc double sourc double nff0		0; k <nno< th=""><th>N 11 11 11</th><th></th><th>l</th><th></th><th></th><th></th><th>3 400</th></nno<>	N 11 11 11		l				3 400
Page 17		- -				· · · · · · · · · · · · · · · · · · ·										MIC
wet9lw.c	t2plus * nf22[nv2[k]]; k] = lw_sources_bulk(2, 2, nv4[k], k, nv2[k],	<pre>tin1 * nf13[k] + tin1plus1 * nf13[nv3[k]] +</pre>	Ę,		<pre>timinus * nf14[nv2[k]] + tlcenter * nf14[k] + tlplus * nf14[nv4[k]]; k] = lw_sources_bulk(1, 4, nv2[k], k, nv4[k],</pre>	Lminus, Licenter, Ligius, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); t2minus * nf24[nv2[k]] + t2center * nf24[k] +	t2plus * nf24[nv4[K]]; [K] = lw_sources_bulk[2, 4, nv2[K], k, nv4[K], t2mluus, t2center, t2plus, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);	<pre>toutlminus2 * nf15[nv7[nv7[k]]] + toutlminus1 * nf15[nv7[k]] + toutlminus1 * nf15[nv7[k]]</pre>	<pre>- IM_Sources_out(1, 5, nv7[nv7[k]], nv7[k], k, toutlminus2, toutlminus1, tout1, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17,</pre>	nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf2/, nf28); tout2minus2 * nf25[nv7[k]] + tout2minus1 * nf25[nv7[k]] + + nf25[nv7[k]] +	<pre>i(k] = lw_sources_out(2, 5, nv7[nv7[k]], nv7[k], k, cottzminus2, toutzminus1, tout2, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);</pre>	<pre>tinl * nf16[k] + tiniplus1 * nf16[nv6[k]] + tiniplus2 * nf16[nv6[nv6[k]]]; k] = lw_sources_in(1, 6, k, nv6[k], nv6[nv6[nv6[k]],</pre>	rini, tintplusi, tintplusa, rint), nf18, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); tin2 * nf26[k] + tin2plus! * nf26[n6[k]] + tin2plus! * nf26[n6[k]] +	<pre>5(k) = lw_sources_in(2, 6, k, nv6[k], nv6[nv6[k]], tin2, tin2plus1, tin2plus2, nt10, nt11, nt12, nt13, nt14, nt15, nt16, nt17, nt18, nt20, nt21, nt22, nt23, nt24, nt25, nt26, nt27, nt28);</pre>	tin1 * nf17[k] + tiniplus1 * nf17[nv7[k]] + tiniplus2 * nf17[nv7[k]]]; k] = lw_sources_in(1, 7, k, nv7[k], nv7[kv]),	tin, tinplus, tinpplus, nil9, nil6, nil6, nil7, nil8, nil0, nil1, nil2, nil3, nil4, nil8, nil0, nil1, nil2, nil3, nil24, nil2, nil6, nil7, nil8), tin2 * nil1() + tin2plus * nil7(nil) + tin2plus * nil7(nil1) + tin2plus * nil7(ni
May 31 1999 19:07	source22[k]	sf13[k] = source13[k]	sf23[k] =	sourcers[K]	sf14[k] = .source14[k]	sf24[k] =	source24[k]	sf15[k] =	source15[k]	sf25[k] =	source25[k]	sf16[k] = source16[k]	sf26[k] =	source26[k]	sf17[k] = source17[k]	sf27[k] = (

tin2, tin2plus1, tin2plus2, n£10, n£11, n£12, n£13, n£14, n£15, n£16, n£17, n£18, n£20, n£21, n£22, n£23, n£24, n£25, n£26, n£27, n£28);	<pre>sfl8[k] = toutlminus2 * nfl8[nv6[k]]] + toutlminus1 * nfl8[nv6[k]] + toutlminus1 * nfl8[nv6[k]] + toutlminus1 * nfl8[k]; sourcess out(1, % nv6[k]), nv6[k], k,</pre>	break; (double sf0[], double sf1[], double sf2[], double sf3[], double sf4[], double sf5[], double source0[], double source1[], double source2[], double source0[], double source4[], double source5[], double source5[], double source7[], double source5[], double notrof[], double nf1[], double nf12[], double nf12[], double nf14[], double nf18[], double nf16[], double nf14[], double nf18[]],	0; K	
	sfl8[sourc	break } void lw (double double doub	{ int k; for (k=0; k <nr< td=""><td></td></nr<>	

1	
.utt.ro	Page 2
тадз.тес	
Printed by soronea from flumags.mec.utt.r	
by sotone	
Printed	ret9fd.c
	5

### ### ### ### #### #### #### ########	<pre> ** wet9fd.c. * wet2fd.c. ** wet2fd.c. ** wet2fd.c. ** include <stdio.h> *include <stdio.ho *="" *include="" <stdio.ho ="" crau="delta_t" ctau="delta_t" cyradx="cyseed" cyradx,="" cyradx_cyrady,="" cyradxs2,="" cyradxy2;="" cyradxy;="" cyrady,="" cyradys2,="" delta_<="" delta_t="" delta_x;="" double="" neq0[],="" neq1[],="" neq5[],="" nff[],="" prodscal;="" tau;="" th=""><th>printf("k", k", k", k", k", k", k", k", k", k",</th></stdio.ho ></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h></pre>	printf("k", k", k", k", k", k", k", k", k", k",
**estdo.h> **estdo.h> **estdo.h> **estdo.h> **estdo.h> **estdo.h> **estdo.h> **estdo.h> **estspead.h* **estspead.gradk.gradks2, cgradks2, cgradky1 **estspead.gradk.gradks2, cgradks2, cgradky2, cgradky1 **estspead.gradk.gradks2, cgradks2, cgradky2, cgradky1 **estspead.gradk.gradks2, cgradks2, cgradky2, cgradky1 **estspead.gradk.gradks2, dalta.k; **estspead.adita.k,	** wet9fd.c ** wet9fead.h ** wet9fead.h ** wet9head.h ** wet9fl, double nffl, double neqfl, double ne	printf("k", "k", "k", "k", "k", "k", "k", "k",
**ecflo.h> **ecflo.h> **ecflo.h> **ecflo.h> **ecflo.h> **ecflo.double mass, double two, double oppeed, double mfffil, double mffil, double mff	#include <stdio.h> #include <stdio.h> #include <stdio.h> #include <stdio.h> #include <stdib.h> #include <stdib.h> #include faction for it is is</stdib.h></stdib.h></stdio.h></stdio.h></stdio.h></stdio.h>	printf("k" k" k" k" k" k" k" k"
**ectio.h>	#include <stdio.h> #include <stdio.h> #include <stdio.h> #include <stdio.h> #include <stdio.h> #include <actio.h. #include="" <actio.h.="" cated.cgradx,="" cated.cgradx<="" cgradx,="" cgradx;="" cgradxs,="" cgradxy;="" deltal,="" deltalx;="" double="" fill,="" neqfl,="" nfill,="" td=""><td>printf("k" h" h" h" h" h" h" h" </td></actio.h.></stdio.h></stdio.h></stdio.h></stdio.h></stdio.h>	printf("k" h" h" h" h" h" h" h"
*** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** *** *** *** *** *** *** *** *** *** *** ** *** *** **	#include <stdio.h> #include <stdio.h> #include <stdio.h> #include fatch.b. #include fatch.h #include math.h #include math.</stdio.h></stdio.h></stdio.h>	* <u>ZZ8Z8Z8Z8</u> Z8
##etShead.hp ##	#include cmath.h> #include cmath.h> #include "wet9head.h" void first_upwind(double mass, double tau, double nf2[],	grangrak "
mass, double tau, double capeed,	#include "wet9head.h" void first_upwind(double mass, double tau, double cspeed, double nf2[], double nf2[], double nf3[], double nf4[], double nf5[], double nf6[], double nf4[], double nf4[], double nf6[], double nf4[], double nf4[], double nf6[], double nf4[], double ctau, cgrad, cgradx, cgrady, cgradxs2, cgradys2, cgradxy; double prodscal; ctau = delta_t / tau; cgrad = cspeed* delta_t / delta_x; cgradx = cspeed* delta_t / delta_x;	9 2 9 2 9 2 9 2 5
######################################	<pre>void first_upwind(double mass, double tau, double cspeed,</pre>	nff0[k] = nff1[k] = cgrad - cgrad nff3[k] = nff3[k] = nff3[k] = cgrad
	double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double nf5[], double neq5[], double nf5[], fouble neq5[], adouble neq5[], funk; double craad, cgrad, cgrady, cgradxs2, cgradys2, cgradxy; double gradx2, cgradx2, cgradx2; double prodscal; ctau = delta_t / tau; ctau = delta_t / tau; cgrad = cspeed * delta_t / delta_x; cgradx = cspeed * delta_t / delta_x; cgradx = cspeed * delta_t / delta_x; cgradx = cspeed * delta_t / delta_x;	
Ca (1) }	double nff3[], double nff5[], double nff3[], double nff5[], double nff6[], double nff7[], double nff8[], double neq0[], double neq1[], double nff8[], double neq0[], double neq1[], double neq2[], double cgradx, cgradx, cgradx, cgradxs, cgradys2, cgradx; double dummy_force, double prodscal; ctau = delta_t / tau; cgrad = cspeed* delta_t / delta_x; cgradx = cspeed* delta_t / delta_x; cgradx = cspeed* delta_t / delta_x;	fglk] = cgrad nff3[k] = cgrad nff4[k] - cgrad
oa (<pre>double neq0[], double neq1[], double neq3[], double neq3[], double neq4[], { int k; double neq6[], double neq4[], double crau, cgradx, cgradx, cgradx, cgradx; cgradx, cgradx2, cgradx2, cgradx2, cgradx2, cgradx2, cgradx2, cgradx2, cgradx2, cgradx4; cqrau = delta_t / tau; crau = delta_t / tau; cgradx = cspeed* delta_t / delta_x; cgradx = cspeed* delta_t / delta_x; cgradx = cgredx delta_x; cgradx delta_x; cgradx; cgradx;</pre>	- cgrad nff4[k] = - cgrad
Call (<pre>f int k; double ctau, cgrad, cgradx, cgradxs2, cgradx92, cgradxy; double cgradx2, cgradxy2; double dummy_force; double prodscal; ctau = delta_t / tau; cgradx = cspeed* delta_t / delta_x; cgradx = cspeed* delta_t / delta_x; cgradx = cspeed* delta_t / delta_v;</pre>	
Ca	<pre>double cgradx2, cgradxy2; double dummy_force; double prodscal; ctau = delta_t / tau; cyradx = cspeed * delta_t / delta_x; cyradx = cspeed * delta_t / delta_x; cyradx = cspeed * delta_t / delta_x;</pre>	mff5[k] = cgrad - cgrad - cgrad
Ca () } }	ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_x; cgradx = cspeed * delta_t / delta_x;	nitolki - cgrad - cgrad
ca 1 1 1 1 1 1 1 1 1	namedy = named * delta V;	- cgrad
oase, nfff nfff nfff nfff nfff nfff nfff nf	gradus = cspeed * sgrt((double) 2) * delta_t / delta_x; cgradus2 = reneed * sgrt((double) 2) * delta_t / delta_v;	nrië(K) = - cgrad - cgrad
nfff nfff nfff nfff nff nff nff nff nff	cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y); cgradx2 = cspeed * delta_t / (2.000/delta_x*);	
nfff nfff nfff nfff nff nff nff	cgradxy2 = cspeed * delta.t / croco caraca,;;; cgradxy2 = cspeed * delta.t / (2.000 * sgrt(delta.x*delta.x + delta.y*delta.y));	77
		niilkj = - cgrac nff2[k] =
nf([k]-neq0[k])*ctau, nf([k]-neq0[k])*ctau, nf([k]-neq1[k])*ctau * (nf([k]-neq2[k])*ctau * (nf2[k]-neq2[k])*ctau * (nf2[k]-neq3[k])*ctau * (nf3[k]-neq3[k])*ctau * (nf4[n-neq4]k])* nf5[k]-neq5[k])*ctau * (nf4[n-neq5[k])*ctau * (nf4[n-neq5[k])*ctau * (nf5[k]-neq5[k])*ctau * (nf5[k]-neq5[k])*ctau * (nf5[k]-neq5[k])*ctau * (nf6[k]-neq6[k])*ctau * (nf6[k]-neq6[k])*ctau * (nf6[k]-neq6[k])*ctau * (nf6[k]-neq6[k])*ctau * (nf6[k]-neq6[k])*ctau * (nf6[k]-neq6[k])*ctau * (nf7[nv2[k]) - nf1[k]) * (nf7[nv2[k]) - nf1[k]) * (nf7[nv2[k]) - nf1[k]) * (nf8[k]-neq8[k])*ctau * (nf8[k]-neq8[k]-neq8[k])*ctau * (nf8[k]-neq8[k]-ne	IOF (K=U) KINOGES_ALL; K++) ewitch/houndary mode(k)	- cgrac nff3[k] =
= nf0[k] - (nf0[k]-neq0[k])*ctau; cdx * cox(1] * (nf1k]-neq0[k])*ctau; cdx * cox(1] * (nf1k]-neq1[k])*ctau = nf2[k] - (nf2[k]-neq2[k])*ctau cdy * coy(2] * (nf2[k] - nf2[nv4[k]]); cdy * coy(2] * (nf2[k] - nf3[k]); cdy * cox(2] * (nf3[k]-neq4[k])*ctau cdx * cox(3] * (nf3[k]-neq4[k])*ctau cdx * cox(4] * (nf4[k]-neq4[k])*ctau cdy * coy(4] * (nf4[k]-neq4[k])*ctau cdy * coy(5] * (nf5[k] - nf5[nv3[k]); cdy * coy(5] * (nf5[k] - nf5[nv3[k]); cdy * coy(5] * (nf5[k] - nf5[nv4[k]]); cdy * coy(5] * (nf6[k]-neq6[k])*ctau cdy * coy(5] * (nf6[k]-neq6[k])*ctau cdy * coy(6) * (nf6[nv1[k]] - nf6[k]); cdy * coy(7] * (nf7[nv2[k]] - nf7[k]); cdy * cox(7] * (nf7[nv2[k]] - nf7[k]); cdy * cox(7] * (nf7[nv2[k]] - nf7[k]); cdy * cox(8] * (nf8[k] - nf8[k])*ctau cdy * cox(8] * (nf8[k] - nf8[k]); c		- cgrac nff4[k] =
full(x) = nil(nvs(x)); fun(x)(x) = nil(nvs(x)); fun(x)(x) = neg(x)(x); fun(x)(x) = neg(x)(x); fun(x)(x)(x) = neg(x)(x); fun(x)(x)(x) = neg(x)(x); fun(x)(x)(x) = neg(x)(x); fun(x)(x) = neg(x); fun(= nf0[k] = nf1[k]	- cgrac nff5[k] =
file line	- ' -	- cgrac nff6[k] =
f4(k]-neq4(k])*ctau	nff3[k] = nf3[k] = (nf3[k] ************************************	- cgrac
## ## ## ## ## ## ## ## ## ## ## ## ##	nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctan - carady * ecy[4] * (nf4[nv2[k]] - nf4[k]);	nff7[k] = - cgrae
	nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau - cgradx * ecx[s] * (nf5[k] - nf3[k]) - cgradx * ecx[s] * (nf5[k] - nf5[m])*	- cgrad nff8[k] -
* (nfc[nt] nfc[nv4[k]); * (nfc[nt] nfc[nv4[k]); * (nf7[nv2[k]] - nf7[k]); * (nf7[nv2[k]] - nf7[k]); * (nf8[k] nf8[k]); * (nf8[k] - nf8[nv3[k]]); * (nf8[nv2[k]] - nf8[nv3[k]]); * (nf8[nv2[k]] - nf8[nv3[k]]); * (nf8[nv2[k]] - nf8[k]);	- cgrady * ecy[5] * (ht5]k] - ht5[ht4]j); hff6[k] = hf6[k] - (hf6[k]) - heg6[k]) * ctau	- cgrad
* (nf7[nv1[k]] - nf7[k]) * (nf7[nv2[k]] - nf7[k]); * (nf8[k]-neg8[k])* (stau * (nf8[k] - nf8[nv3[k]]) * (nf8[nv2[k]] - nf8[k]); *g %g %g %g %g %g %g %g %g	- Ggradx * ecx[6] * (nfo[n1.k]) - cgrady * ecx[6] * (nfo[x] - nfo[nv4[k]]); nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau	1
ecx[8] * (nf8[nv2[k]] - nf8[k]); ecy[8] * (nf8[nv2[k]] - nf8[k]); ecy[8] * (nf8[nv2[k]] - nf8[k]);	* (nf7[nv1[k]] * (nf7[nv2[k]]	<u>-</u>
हैते हैते हैते हैते हैते हैते हैते हैते	nistkj - (nistkj-negotkj) k * ecx[8] * (nf8[k] - nf8 / * ecy[8] * (nf8[nv2[k]]	
	දීට දීට දීට දීට දීට දීට දීට දීට දීට	

neq0[k],nee neq6[k],nee k,nec[k],ne nf6[k],nf7[f,k,nff0[k],nf7[k,nff0[k],nf7[nff6[k],nf7[<pre>case 3:</pre>	<pre>case 4: /* right wall */ nff1kl = nf0[k] - (nf0[k]-neq0[k])*ctau; nff1kl = nf1[k] - (nf1k]-neq1[k])*ctau - cgradx * ecx[1] * (nf1k] - nf1[m3[k]]); nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau - cgradx * ecx[2] * (nf2[k] - nf2[m4[k]]); nff3[k] = nf3[k] - (nf3[k] - nf2[k]) * (nf3[k] - nf2[k]); nff4[k] = nf4[k] - (nf3[k] - nf3[k]); nff4[k] = nf5[k] - (nf3[k] - nf4[k]); nff5[k] = nf5[k] - (nf5[k] - nf5[m3[k]); - cgradx * ecx[3] * (nf5[k] - nf5[m3[k]); nff6[k] = nf6[k] - (nf5[k] - nf5[m3[k]); - cgrady * ecy[5] * (nf5[k] - nf5[m3[k]); nff6[k] = nf6[k] - (nf6[k] - nf6[k]); - cgrady * ecy[6] * (nf6[k] - nf6[m3[k]); nff7[k] = nf7[k] - (nf7[k] - nf7[k]); - cgrady * ecy[7] * (nf7[k] - nf7[k]); - cgrady * ecy[8] * (nf8[k]] - nf7[k]); - cgrady * ecy[8] * (nf8[k]] - nf8[k]); - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]); - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]]; - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]]; - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]]; - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]]; - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]]; - cgrady * ecy[8] * (nf8[k]] - nf8[k]] - nf8[k]]; - cgra</pre>

wet9is.c

utt.ro
3.тес
flumag
from
sofonea
by
Printed

##
ecprod[[6]; ecprod[[0]-prodecal]*neq10 ecprod[[1]-prodecal]*neq11 ecprod[[1]-prodecal]*neq10 ecprod[[3]-prodecal]*neq14 ecprod[[3]-prodecal]*neq14 ecprod[[6]-prodecal]*neq16 ecprod[[6]-prodecal]*neq18 ecprod[[1]-prodecal]*neq20 ecprod[[1]-prodecal]*neq20 ecprod[[3]-prodecal]*neq20 ecprod[[3]-prodecal]*neq20 ecprod[[3]-prodecal]*neq20 ecprod[[3]-prodecal]*neq20 ecprod[[6]-prodecal]*neq20 ecprod[[6
* uxloc[k]; ecprod1[0]-prodscal]*neq10 ecprod1[1]-prodscal]*neq10 ecprod1[3]-prodscal]*neq13 ecprod1[3]-prodscal]*neq14 ecprod1[3]-prodscal]*neq14 ecprod1[6]-prodscal]*neq16 ecprod2[1]-prodscal]*neq16 ecprod2[1]-prodscal]*neq20 ecprod2[1]-prodscal]*neq20 ecprod2[3]-prodscal]*neq26 ecprod2[3]-prodscal]*neq26 ecprod2[3]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[6]-prodscal]*neq28 ecprod2[7]-prodscal]*neq28 ecprod2[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq28 ecprod3[8]-prodscal]*neq38 ecprod3[
* uxloc(k); ecprod1[0]-prodscal)*neq10 ecprod1[1]-prodscal]*neq13 ecprod1[2]-prodscal]*neq13 ecprod1[3]-prodscal]*neq13 ecprod1[6]-prodscal]*neq13 ecprod1[6]-prodscal]*neq14 ecprod1[6]-prodscal]*neq16 ecprod2[1]-prodscal]*neq26 ecprod2[1]-prodscal]*neq26 ecprod2[1]-prodscal]*neq26 ecprod2[2]-prodscal]*neq26 ecprod2[3]-prodscal]*neq26 ecprod2[6]-prodscal]*neq26 ecprod2[
ecprodi[7] -prodecal] *neq17 ecprodi[8] -prodecal] *neq21 ecprod2[1] -prodecal] *neq20 ecprod2[1] -prodecal] *neq20 ecprod2[2] -prodecal] *neq20 ecprod2[3] -prodecal] *neq20 ecprod2[3] -prodecal] *neq20 ecprod2[5] -prodecal] *neq20 ecprod2[5] -prodecal] *neq20 ecprod2[6] -prodecal] *neq20 ecprod2[6] -prodecal] *neq20 ecprod2[8] +
k]] + iscenter1 * ff11[k] k]] + iscenter1 * ff12[k] k]] + iscenter1 * ff13[k] k]] + iscenter1 * ff14[k] k]] + iscenter1 * ff15[k] k]] + iscenter1 * ff15[k] k]] + iscenter1 * ff16[k] k]] + iscenter1 * ff17[k] k]] + iscenter1 * ff17[k] k]] + iscenter1 * ff18[k] k]] + iscenter1 * ff18[k]
k]] + iscenter1 * ff11[k] k]] + iscenter1 * ff12[k] k]] + iscenter1 * ff13[k] k]] + iscenter1 * ff14[k] k]] + iscenter1 * ff14[k] k]] + iscenter1 * ff15[k] k]] + iscenter1 * ff15[k] k]] + iscenter1 * ff17[k] k]] + iscenter1 * ff18[k] k]] + iscenter1 * ff18[k] k]] + iscenter1 * ff18[k] k]] + iscenter1 * ff18[k]
<pre>k[]; k[]; k[]; k[]; k[]; k[]; k[]; k[];</pre>

Jun 20 1999 18:01	wet9is.c	Page 2
£22[k]	= isminus2 * ff22[nv4[k]] + iscenter2 * ff22[k] +	
f23[k]		
£24[k]	* *	
£25[k]	<pre>= ispluse</pre>	
£26[k]	* *	
£27[k]	* isminus2 *	
£28[k]	= isminus2	
break;	N 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
case 1:		
£10[k]	= ff10[k]; = isminus1 '	
£12[k]	<pre>isplus1 * ff11[nv1[k]]; = tin1 * ff12[k] + tin1plus1 * ff12[nv2[k]] +</pre>	
£13[k]	= isminus1 =	
£14[k]	<pre>isplus1 * iiislnv3(k)]; = toutlminus2 * ff14[nv2[nv2[k +oottlminus1 * ff14[nv2(v)] +</pre>	
f15[k]	concinitions I result to the tinity of the tinity of the tinity of the tinity of the tinitions of the tinition of the tinitions of the tinition of the tinitions of the tinition of the tinit	
£16[k]	= tinl * ff16[k] + tinlplus1 + tinlplus1 + tinlplus1	
£17[k]	_ toutlminus2 * ff17[nv5[nv5]k +outlminus1 * ff17[nv5[k]] +	
f18[k]	= toutiminus; # ff18[nv6[nv6[k]]] + tout1minus1 * ff18[nv6[k]]] + tout1minus1 * ff18[nv6[k]] + tout1 * ff18[k]	
f20[k]	= ff20[k] = isminus	
f22 [k]	<pre>isplus2 * ff21[nv1[k]]; = tin2 * ff22[k] + tin2plus1 * ff22[nv2[k]] +</pre>	
£23[k]	11	
£24[k]	<pre>ispius2 * ii25[nv3[k]]; = tout2minus2 * if24[nv2[nv2[k]] + tout2minus1 * ff24[nv2[k]] +</pre>	
f25[k]	= tin2 * ff25[k] + tin2plus1 *	
£26[k]	= tin2 * ff26[k] + tin2plus1 * ff26[nv6[k]] + tin2plus1 * ff26[nv6[k]] + tin2plus2 * ff26[nv6[nv6[k]]]	
£27[k]	= tout2minus2 * ff27[nv5[k]] + tout2minus1 * ff27[nv5[k]] + tout2 * ff27[k];	
£28[k]	<pre>= tout2minus2 * ff tout2minus1 * ff</pre>	
break; case 2:		
£10[k] £11[k]		
f12[k]	isplus1 * ifil[nvl[k]]; = toutliminus2 * ifil2[nvv[k]]] +	
£13[k]	= isminus1 * ff13[nv1[k]] + i isplus1 * ff13[nv3[k]];	
£14[k]		
£15[k]	= toutlminus2 * ff15[nv7[k]]] + toutlminus1 * ff15[nv7[k]] + tout1 * ff15[k];	

wet9is.c

tt.ro	
3.mec.ul	
lumaga	
from f	
ofonea	
d by s	
Printe	

wet9is.c

Page 3 Jun 20 1999 18:01	f10[k] f11[k] f12[k]	113[K]	f17(K) f18(K) f20(K) f22(K) f22(K) f22(K)	£24[k]	£26[k]	128[K] break; }	double centl(double	double cent2(double { return (isminus2) } double fin1(double { return (tin1 * n1	double fin2 (double { return (tin2 * n) } double out1(double
Jun 20 1999 18:01 wet91s.c	<pre>f16[k] = toutlminus2 * ff16[nv8[k]]] +</pre>	<pre>f20[k] = ff20[k]; f21[k] = iaminus2 * ff21[nv3[k]] + iscenter2 * ff21[k] +</pre>	<pre>isplus2 * ff23[nv3[k]]; ff24[k] = tin2 * ff24[k] + tin2plus1 * ff24[nv4[k]] + ff25[k] = tout2minus2 * ff25[nv7[nv7[k]]] +</pre>	break;	<pre>f10(k) = ff10(k); f11[k] = tin1 * ff11[k] + tin1plus1 * ff11[nv1[k]] +</pre>	<pre>isplus1 * ffl2[nv2[k]]; f13[k] = toutlminus2 * ff13[nv1[k]]] + toutlminus1 * ff13[nv1[k]] + tout1 * ff13[k]; f14[k] = isminus1 * ff14[nv2[k]] + iscenter1 * ff14[k] +</pre>	= toutlm toutlm = toutlm toutlm = tin1 *	<pre>f20[k] = ff20[k]; f21[k] = tin1 * ff21[k] + tin1plus1 * ff21[nv1[k]] +</pre>	isplus1 * ff24[nv4[k]]; tin1 * ff25[k] + tin1plus1 * ff25[nv5[k]] + tin1plus2 * ff25[nv5[k]] + toutlminus2 * ff26[nv8[k]] + tout1 * ff26[k]; toutlminus1 * ff26[nv8[k]] + tout1 * ff26[k]; toutlminus2 * ff27[nv5[k]] + tout1 * ff27[k]; tin1 * ff28[k] + tin1plus1 * ff28[nv8[k]] +

<pre>f10(k) = f10(k); f11(k) = cottlminus2 * ff11(nv3[nv3[k]]) +</pre>
* ff21[nv3[nv3[k]]] + * ff21[nv3[h]] + tottl * ff21[k] ff22[nv2[k]]; [k] + tinplus1 * ff23[nv3[k]] + tinplus2 * ff23[nv3[k]] + tinplus2 * ff23[nv3[k]] + tinplus2 * ff23[nv3[k]] + ff24[nv4[k]]; * ff25[nv7[nv7[k]]] + 1scenter1 * ff24[k] + * ff25[nv7[nv7[k]]] + tinplus1 * ff26[nv6[k]] + tinplus1 * ff26[nv6[nv6[k]] + tinplus2 * ff26[nv6[k]] + tinplus2 * ff26[nv6[k]] + tinplus2 * ff26[nv6[k]] + tinplus2 * ff27[nv7[k]] + tinplus2 * ff27[nv7[k]] + tinplus2 * ff27[nv7[k]] + tinplus2 * ff27[nv7[nv7[k]] + tinplus2 * ff27
<pre>double cent1(double n1, double n2, double n3) { return (isminus1 * n1 + iscenter1 * n2 + isplus1 * n3); } double cent2(double n1, double n2, double n3)</pre>
<pre>freturn (isminus2 * n1 + iscenter2 * n2 + isplus2 * n3); } double fin1(double n1, double n2, double n3) freturn (tin1 * n1 + tin1plus1 * n2 + tin1plus2 * n3); }</pre>
<pre>double fin2(double n1, double n2, double n3) { return (tin2 * n1 + tin2plus1 * n2 + tin2plus2 * n3); } double out1(double n1, double n2, double n3) { return (uplminus2 * n1 + uplminus1 * n2 + upl * n3); } double out2(double n1, double n2, double n3) { }</pre>

return (up2minus2 * n1	+ up2minus1 * n2 + up2 * n3);	
void islb_upwind(void)		
<pre>int k; double prodscal; for(k=0; k<nnodes_all;< pre=""></nnodes_all;<></pre>	.all; k++)	
prodscal = csforce	x * uxloc[k] + csforce_y * uxloc[k];	
ff10(k) = f10(k) = f21(k) = f21(k) = f212(k) = f212(k) = f213(k) =	-ctaul*(f10[k]-neq10[k])+(ecprod1[0]-prodscal)*neq10[k]; -ctaul*(f11[k]-neq11[k])+(ecprod1[1]-prodscal)*neq12[k]; -ctaul*(f12[k]-neq13[k])+(ecprod1[3]-prodscal)*neq13[k]; -ctaul*(f13[k]-neq13[k])+(ecprod1[3]-prodscal)*neq13[k]; -ctaul*(f13[k]-neq13[k])+(ecprod1[4]-prodscal)*neq14[k]; -ctaul*(f15[k]-neq15[k])+(ecprod1[6]-prodscal)*neq16[k]; -ctaul*(f16[k]-neq16[k])+(ecprod1[6]-prodscal)*neq16[k]; -ctaul*(f18[k]-neq17[k])+(ecprod1[7]-prodscal)*neq18[k]; -ctaul*(f18[k]-neq17[k])+(ecprod1[8]-prodscal)*neq18[k];	
ff20[k] = f20[k]- ff21[k] = f21[k]- ff22[k] = f22[k]- ff23[k] = f22[k]- ff23[k] = f24[k]- ff25[k] = f24[k]- ff26[k] = f25[k]- ff26[k] = f25[k]- ff27[k] = f27[k]-	-ctau2* (f20[k]-neq20[k]) + (ecprod2[0]-prodscal) *neq20[k]; -ctau2* (f22[k]-neq22[k]) + (ecprod2[1]-prodscal] *neq21[k]; -ctau2* (f22[k]-neq22[k]) + (ecprod2[2]-prodscal] *neq22[k]; -ctau2* (f23[k]-neq24[k]) + (ecprod2[3]-prodscal] *neq34[k]; -ctau2* (f26[k]-neq24[k]) + (ecprod2[5]-prodscal] *neq34[k]; -ctau2* (f26[k]-neq26[k]) + (ecprod2[6]-prodscal] *neq34[k]; -ctau2* (f26[k]-neq36[k]) + (ecprod2[6]-prodscal] *neq36[k]; -ctau2* (f26[k]-neq36[k]) + (ecprod2[7]-prodscal] *neq36[k]; -ctau2* (f28[k]-neq36[k]) + (ecprod2[8]-prodscal] *neq37[k];	
for(k=0; k <nnodes_all; k++)<="" td=""><td>k++)</td><td></td></nnodes_all;>	k++)	
<pre>{ switch (boundary_mode[k])</pre>	ode[k])	
Case 0: f10[k] = ff10[f11[k] = out! f12[k] = out! f13[k] = out! f14[k] = out! f16[k] = out! f16[k] = out! f17[k] = out! f17[k] = out!	<pre>ff10(k); out1(ff11[nv3[nv3[k]]),ff11[nv3[k]],ff11[k]); out1(ff12[nv4[nv4[k]]),ff13[nv1[k]],ff12[k]); out1(ff13[nv1[nv1[k]]],ff13[nv1[k],ff13[k]); out1(ff14[nv2[nv2[k]]],ff14[nv2[k]],ff14[k]); out1(ff16[nv8[nv8[k]]),ff15[nv7[k]],ff18[k]); out1(ff16[nv8[nv8[k]]),ff16[nv8[k]],ff18[k]); out1(ff18[nv8[nv8[k]]),ff17[nv5[k]],ff17[k]);</pre>	
f20 [k] = ff20 [f21 k] = out2 [f22 k] = out2 [f23 k] = out2 [f24 k] = out2 [f25 k] = out2 [f25 k] = out2 [f28 k]	10 [k]; 12 (ff21[nv3[nv3[k]]), ff22[nv3[k]], ff22[k]); 12 (ff22[nv4[nv4[k]]), ff23[nv1[k]], ff23[k]); 12 (ff23[nv4[nv1[k]]), ff23[nv1[k]], ff23[k]); 12 (ff24[nv2[nv2[k]]), ff24[nv2[k]), ff23[k]); 12 (ff25[nv7[nv7[k]]), ff25[nv8[k]], ff25[k]); 12 (ff25[nv7[nv7[k]]), ff25[nv8[k]], ff25[k]); 12 (ff28[nv6[nv8[k]]), ff28[nv6[k]], ff28[k]); 12 (ff28[nv6[nv6[k]]), ff28[nv6[k]], ff28[k]);	
Case 11: ff10(f110(R) = out.(f112(R) = out.(f13(R) = out.(f13(R) = out.(f15(R) = out.(f16(R) = out.(f16(R) = out.(f16(R) = out.(= ff10[k]; = out1(ff11[nv3[nv3[k]]),ff11[nv3[k]],ff11[k]); = out1(ff13[nv1[k]]),ff13[nv1[k]]); = out1(ff13[nv1[k]]),ff13[nv1[k]],ff13[k]); = out1(ff14[nv2[nv2[k]]],ff14[nv2[k]],ff14[k]); = cent1(ff18[nv2[k]],ff16[k],ff16[nv5[k]]); = out1(ff18[nv8[k]],ff16[k],ff16[nv6[k]]); = out1(ff18[nv8[nv5[k]],ff18[nv6[k]],ff18[nv6[k]]);	

20 1999 18:01	wet9is.c	Page 6
f20 [k] = f21 [k] = f22 [k] = f23 [k] = f24 [k] = f25 [k] = f26 [k] = f28 [k] = break;	= ff20[k]; = cut2[ff2[lnv3[k]]],ff22[k],ff22[nv3[k]]), = cut2[ff22[nv4[k]]],ff22[k],ff22[nv]]); = cut2[ff23[nv1[k]]],ff23[k],ff23[k]]); = cut2[ff23[nv2[k]],ff24[nv2[k]]); = cent2[ff25[nv7[k]],ff25[k],ff25[nv5[k]]); = cent2[ff26[nv8[k]],ff26[k],ff26[nv6[k]]); = cut2[ff27[nv5[nv5]]],ff28[k],ff26[nv6[k]]); = cut2[ff28[nv6[nv5]]],ff28[nv6[k]]);	
Case 22: f10 [k] f11 [k] f13 [k] f14 [k] f16 [k] f16 [k] f17 [k]	= ff10[k]; = out1(ff11[nv3[nv3[k]]),ff11[nv3[k]],ff11[k]); = out1(ff12[nv4[nv4[k]]),ff12[nv4[k]],ff12[k]); = out1(ff13[nv1[nv1[k]]),ff13[nv1[k]],ff13[k]); = out1(ff14[nv2[k]]),ff14[k],ff14[nv4[k]]); = out1(ff16[nv7[nv7[k]]),ff16[nv4[k]]); = out1(ff18[nv8[nv8[k]]),ff16[nv8[k]]); = cent1(ff18[nv8[k]],ff18[k],ff17[nv7[k]]);	
f20 [k] f21 [k] f22 [k] f23 [k] f24 [k] f25 [k] f26 [k] f28 [k] break;	= ff20[k]; = out2(ff22[nv3[nv3[k]]),ff21[nv3[k]],ff22[k]); = out2(ff22[nv4[nv4[k]]),ff22[nv4[k]],ff22[k]); = out2(ff23[nv1]nv1[k]]),ff23[nv1]k]],ff23[nv1]; = out2(ff24[nv2[k]],ff24[k],ff24[nv4[k]]); = out2(ff25[nv7[nv7[k]]),ff25[nv7[k]],ff25[k]); = out2(ff25[nv7[nv7[k]]),ff25[nv8[k]],ff26[k]); = cont2(ff26[nv8[k]],ff26[nv8[k]]); = cont2(ff28[nv6[k]],ff28[k]),ff28[nv8[k]]);	
Case 1: f10[k] f11[k] f12[k] f13[k] f14[k] f15[k] f16[k] f17[k]	<pre>ff10(k); out1(ff11[nv3[nv3[k]],ff11[nv3[k]],ff11[k]); fin1(ff12[k],ff12[nv2[k]],ff12[nv2[nv2[k]]]); out1(ff13[nv1[nv2[k]]),ff13[nv1[k]]); out1(ff14[nv2[kv]]),ff14[nv2[k]],ff14[kv]); fin1(ff14[nv2[kv]]),ff14[nv2[k]],ff14[nv2[k]]); = fin1(ff12[kv],ff15[nv5[kv]]),ff17[nv5[k]]); = out1(ff17[nv5[nv5[kv]]),ff17[nv5[k]]); = out1(ff17[nv5[nv5[kv]]),ff17[nv5[k]]);</pre>	
120 [k] 121 [k] 122 [k] 123 [k] 124 [k] 125 [k] 126 [k] 128 [k] 128 [k]	<pre>ff20(k]; out2(ff21[ny3[ny3[k]]],ff21[ny3[k]],ff21[k]); e inA(ff22[k],ff22[ny2[ny2[ny2[k]]]); out2(ff22[k],ff22[ny2[k]],ff23[k]); out2(ff23[ny1[k]]),ff23[ny1[k]],ff24[k]); e inA(ff25[k],ff25[ny5[k]],ff24[ny2[k]],ff24[k]); e inA(ff25[k],ff25[ny5[k]],ff25[ny5[ny5[ny5[k]]]); e inA(ff25[k],ff3[ny6[k]],ff25[ny6[k]]); e out2(ff28[ny6[ny5[k]]],ff28[ny6[k]]);</pre>	
Case 2: f10[k] f11[k] f11[k] f13[k] f14[k] f15[k] f16[k] f17[k]	<pre>ff10[k]; out1(ff11[nv3[nv3[k]]),ff11[nv3[k]],ff11[k]); = out1(ff22[nv4[nv4[k]]),ff12[nv4[k]]),ff12[k]); = out1(ff13[nv1[nv1]]),ff13[nv1[k]],ff13[k]); = fin1[ff14[k],ff14[nv4[k]],ff14[nv4[nv4[nv4[k]]]); = out1(ff15[nv7[nv7[k]]),ff15[nv7[k]],ff18[k]); = fin1(ff17[k],ff18[nv3[k]],ff18[nv8[nv3[k]]); = fin1(ff17[k],ff18[nv3[k]],ff18[nv3[k]]);</pre>	
£20[k] £21[k] £22[k]	<pre>= ff20(k]; = out2(ff21[nv3[nv3[k]]),ff21[nv3[k]],ff21[k]); = out2(ff22[nv4[nv4[k]]),ff22[nv4[k]]);</pre>	

	_
3.mec.utt.ro	Page 8
rinted by sofonea from flumag3.mec.utt.ro	\$ 5
y sofonea f	
Printed b	wet9is.c

Jun 20 1999 18:01	Case 1: E11 [K]	
Jun 20 1999 18:01 Page 7	<pre>f23(k] = out2(ff23[nv1[nv1[k]]),ff23[nv1[k]]; f24(k] = fin2(ff24[k],ff24[nv4[k]],ff24[nv4[k]]]; f25(k] = out2(ff26[nv7[nv7[k]],f25[nv7[k]],f25[k]); f25(k] = out2(ff26[nv8[nv8[k]]),ff25[nv8[k]]); f27(k] = fin2(ff28[k],ff27[nv7[k]],ff28[nv8[nv8[k]]]); break; } } </pre>	first Richard (Note)

<pre>case 1: f10[k] = ff10[k]; f11[k] = cf11 * ff11[nv3[k]] + oneminus1 * ff11[k]; f12[k] = oneplus1 * ff12[k] - cf11 * ff12[nv2[k]]; f13[k] = cf11 * ff13[nv1[k]] + oneminus1 * ff13[k]; f14[k] = cf11 * ff13[nv2[k]] + oneminus1 * ff13[k]; f15[k] = oneplus1 * ff15[k] - cf11 * ff15[nv5[k]]; f17[k] = cneplus1 * ff15[k] - cf11 * ff15[nv5[k]]; f17[k] = cf11 * ff17[nv5[k]] + oneminus1 * ff17[k];</pre>	f20[k] = ff10[k]; f21[k] = cf12 * ff21[nv3[k]] + oneminus2 * ff21[k]; f22[k] = oneplus2 * ff22[k] - cf12 * ff22nv2[k]]; f23[k] = cf12 * ff23[v1[k]] + oneminus2 * ff23[k]; f24[k] = cf12 * ff24[v1[k]] + oneminus2 * ff24[k]; f25[k] = oneplus2 * ff25[k] - cf12 * ff25[nv5[k]]; f26[k] = oneplus2 * ff25[k] - cf12 * ff25[nv6[k]]; f27[k] = cf12 * ff27[nv5[k]] + oneminus2 * ff27[k]; break;	<pre>case 2: f10[k] = ff10[k]; f11[k] = cril * ff11[nv3[k]] + oneminus1 * ff11[k]; f12[k] = cril * ff12[nv4[k]] + oneminus1 * ff12[k]; f13[k] = cril * ff13[nv1[k]] + oneminus1 * ff13[k]; f14[k] = orfl * ff13[nv1[k]] + oneminus1 * ff13[k]; f15[k] = cril * ff15[k] + orfl * ff15[k]; f15[k] = cril * ff15[k] + oneminus1 * ff15[k]; f17[k] = oreplus1 * ff17[k] + oreminus1 * ff16[k]; f17[k] = oneplus1 * ff17[k] - cf11 * ff18[nv8[k]]; f18[k] = oneplus1 * ff18[k] - cf11 * ff18[nv8[k]];</pre>	f20[k] = ff20[k]; f21[k] = cf12 * ff21[nv3[k]] + oneminus2 * ff21[k]; f22[k] = cf12 * ff22[nv4[k]] + oneminus2 * ff22[k]; f23[k] = cf12 * ff22[nv1[k]] + oneminus2 * ff22[k]; f24[k] = oneplus2 * ff24[k] - cf12 * ff24[nv4[k]]; f25[k] = cf12 * ff25[nv7[k]] + oneminus2 * ff25[k]; f26[k] = cf12 * ff25[nv7[k]] + oneminus2 * ff26[k]; f27[k] = oneplus2 * ff27[k] - cf12 * ff27[nv7[k]]; f28[k] = oneplus2 * ff28[k] - cf12 * ff28[nv8[k]];	break; case 3:	f10[k] = ff10[k]; f11[k] = oneplus1 * ff11[k] - cf11 * ff11[nv1[k]]; f12[k] = cf11 * ff12[nv4[k]] + oneminus1 * ff12[k]; f13[k] = cf11 * ff13[nv1[k]] + oneminus1 * ff13[k]; f14[k] = cf11 * ff14[nv2[k]] + oneminus1 * ff14[k]; f15[k] = oneplus1 * ff15[k] - cf11 * ff16[k]; f15[k] = cf11 * ff16[nv8[k]] + oneminus1 * ff16[k]; f17[k] = cf11 * ff16[nv8[k]] + oneminus1 * ff16[k]; f17[k] = cf11 * ff16[nv8[k]] - cf11 * ff16[k];	f20[k] = ff20[k]; f21[k] = oneplus2 * ff21[k] - cf12 * ff21[nv1[k]]; f22[k] = cf12 * ff22[nv1[k]] + oneminus2 * ff22[k]; f23[k] = cf12 * ff23[nv1[k]] + oneminus2 * ff23[k]; f24[k] = cf12 * ff24[nv1[k]] + oneminus2 * ff24[k]; f25[k] = oneplus2 * ff25[k] - cf12 * ff25[k]; f26[k] = cf12 * ff25[k] + oneminus2 * ff26[k]; f27[k] = cf12 * ff25[nv5[k]] + oneminus2 * ff26[k];
---	--	--	--	----------------	---	--

_
ff.70
نب
_
73
<u>س</u>
≊
3.mec.1
ഇ
agg
~
⊑
Ξ
E
O
₽
Ø
Φ
⊑
sofone
0
S
≥
Ω
0
ற
ਵ
Ξ
D.

Jun 20 1999 18:01	wet9is.c	Page 9
break; case 4:		
f10[k] = ff10[k]; f11[k] = cf11 * ff11[nv1[3]] · f12[k] = cf11 * ff21[nv4[k]] · f13[k] = oneplus1 * ff13[k] · f15[k] = cf11 * ff15[nv7[k]] · f15[k] = oneplus1 * ff15[k] · f16[k] = oneplus1 * ff16[k] · f17[k] = oneplus1 * ff17[k] · f18[k] = cf11 * ff18[nv6[k]] ·	ff10[k]; cf11 * ff11[nv1[3]] + oneminus1 * ff11[k]; cf11 * ff12[nv4[k]] + oneminus1 * ff12[k]; oneplus1 * ff13[k] - cf11 * ff13[nv3[k]]; cf11 * ff14[nv2[k]] + oneminus1 * ff14[k]; cf11 * ff15[nv7[k]] + oneminus1 * ff14[k]; oneplus1 * ff16[k] - cf11 * ff16[nv6[k]]; cf11 * ff18[nv6[k]] + oneminus1 * ff18[k];	
f20[k] = ff20[k]; f21[k] = cf12 * ff21[nv1[3]] + f22[k] = cf12 * ff22[nv4[k]] + f23[k] = oneplus2 * ff23[k] - f24[k] = cf12 * ff24[nv2[k]] + f25[k] = cf12 * ff25[nv7[k]] + f25[k] = oneplus2 * ff25[k] - f27[k] = oneplus2 * ff25[k] - f28[k] = cf12 * ff25[k] +	[3]] + oneminus2 * ff21[k]; k] + oneminus2 * ff22[k]; k] + oneminus2 * ff23[k]; k] + oneminus2 * ff24[k]; k] + oneminus2 * ff24[k]; k] + oneminus2 * ff26[k]; k] - cf12 * ff26[nv6[k]]; k] + oneminus2 * ff28[k];	
break;		
,		
void iprop(void)		
int k; double prodscal;		
<pre>for (k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>		
prodscal = csforce_x * uxloc[k]	+ csforce_y * uyloc[k];	
	neq17[k]; (f18[k] - neq18[k]) + neq18[k];	
[k] - ctau2 * - prodscal) * [k] - ctau2 * - prodscal) * [k] - ctau2 *	(f20[k] - neq20[k]) + neq20[k]; (f21[k] - neq21[k]) + neq21[k]; (f22[k] - neq22[k]) +	
odz[2] - prodscal) * = f23[k] - ctau2 * odz[3] - prodscal) * = f24[k] - ctau2 * odz[4] - prodscal) *	degarki) (f23kk) neq23(k); neq23(k); neq24(k);	
[k] - ctau2 * - prodscal) *	(£25[k] - neq25[k]) + neq25[k];	

Jun 20 1999 18:01	wet9is.c	Page 10
<pre>ff26[k] = f26[k] - ctau2 ' (ecprod2[6] - prodscal) ff27[k] = f27[k] - ctau2 ' (ecprod2[7] - prodscal) ff28[k] = f28[k] - crau2 ' (ecprod2[8] - prodscal)</pre>	* (f26[k] - neq26[k]) +) * neq26[k]; * (f27[k] - neq27[k]) +) * neq27[k]; * (f28[k] - neq28[k]) +) * neq28[k];	
/* if(k==0) printf("iter=%d k=%d printf("iter=%d k=%d fl0[k]-neq10[k]); */	ff10=%e ff20=%e\n",iter,k.ff10 k],ff20[k]); neq10-%e f10-neq-%e\n",iter,k,neq10[k],	
<pre>if(key_init > 2) {</pre>		
printf("iter=%d k=% iter,k,gradn1 ff10[k] += kforce *	<pre>#e gradn1y=%e gradn2x=%e y[k],gradn2x[k],gradn2y[k]) * (ext][0] - uxloc[k]) + * (ext][0] - uxloc[k]) +</pre>	\n",
+= kforce	(ecy1[0] - uyloc[k])) (ecx1[1] - uxloc[k]) + (ecy1[1] - uyloc[k]))	
<pre>ff12[k] += kforce * ff13[k] += kforce *</pre>	* (ecx1[2] - uxloc[k]) + * (ecy1[2] - uyloc[k])) * neq12[k] * (ecy1[3] - uyloc[k]) +	
+= kforce	k] * (ecy1[3] - uyloc[k]) * k] * (ecx1[4] - uxloc[k]) +	
ff15[k] += kforce *	* (ecy1[4] - uyloc[k])) * * (ecx1[5] - uxloc[k]) +	
ff16[k] += kforce *	* (ecyl[5] - uyloc[K])) * * (ecx1[6] - uxloc[K]) + * (ecx1[6] - uxloc[K]) *	
ff17[k] += kforce *	* (ecy1[7] - uyloc[k]) + (ecx1[7] - uxloc[k]) + (ecv1[7] - uvloc[k]))	
ff18[k] += kforce *] - uxloc[k]) +] - uyloc[k])) * neq18[k]	
ff20[k] += kforce *	(gradnik[k] * (ecx2[0] - uxloc[k]) +	
ff21[k] += kforce *	c] * (ecx2[1] - uxloc[k]) + (ecx2[1] - uxloc[k]) +	
ff22[k] += kforce *		
ff23[k] += kforce *	[K] * (ecyz[z] - uyloc[K])) * inegzz[K] [K] * (eczz[z] - uyloc[K]) + [V] * (eczz[z] - uwloc[K]) * negzz[V]	
ff24[k] += kforce *	t] * (ecy2[4] - uyloc[k]) + (ecx2[4] - uxloc[k]) +	
ff25[k] += kforce *	* (ecx2[5] - uxloc[k]) + * (ecx2[5] - uxloc[k]) +	
ff26[k] += kforce *	c] * (ecx2[6] = uxloc[k]) + c] * (ecx2[6] = uxloc[k]) +	
ff27[k] += kforce *	* (ecx2[7] - uxloc[k]) + * (ecx2[7] - uxloc[k]) +	
ff28[k] += kforce *	[8] - uxloc[k]) + neq28[k] [8] - uyloc[k]) + neq28[k]	
, , ,		
void ic(void)		
<pre>int k; for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>		
{ switch(boundary_mode[k])		
case 0:		

49

ag3.mec.utt.ro	Page 12
Printed by sofonea from flumag3.mec.utt.ro	wet9is.c

Jun 20 1999 18:01	wet9is.c	Page 11	Jun 20 1999 18:01
f10[k] = ff10[k];			ff23[m
<pre>f11(k) = ff11(k) * nc10[1] + ff1[nv1(k]] * nc11[1] + ff1 ff11[nv3[k]] * nc13[1] + ff1 ff11[nv5(k]] * nc15[1] + ff1 ff11[nv7(k]] * nc17[1] + ff1</pre>	<pre>ffl[[nv2[k]] * nc12[1] + ffl[[nv4[k]] * nc14[1] + ffl[[nv6[k]] * nc16[1] + ffl[[nv8[k]] * nc18[1];</pre>		### ##################################
f12(K) = ff12(K) * nc10[2] + ff12[2[nv2[k]] * nc12[2] + 2[nv4[k]] * nc14[2] + 2[nv6[k]] * nc16[2] + 2[nv8[k]] * nc18[2];		f25[k] = f1 f25[m ff25[m ff25[m ff25[m
f13[K] = ff13[K] * nc10[3] + ff13[ff13[nv1[K]] * nc11[3] + ff13[ff13[nv2[K]] * nc15[3] + ff13[ff13[nv7[K]] * nc17[3] + ff13[nv7[K]] * nv7[K] * nv7[K	3[nv2[k]] * nc12[3] + 3[nv4[k]] * nc14[3] + 3[nv6[k]] * nc16[3] + 3[nv8[k]] * nc18[3];		f26[k] = f1 ff26[m ff26[m ff26]m ff26[m
<pre>f14(k) = ff14(k) * nc10(4) + ff14(nv1(k)) * nc11(4) + ff1 ff14(nv2(k)) * nc13(4) + ff1 ff14(nv5(k)) * nc15(4) + ff1 ff14(nv7(k)) * nc15(4) + ff1</pre>	ff14[nv2[k]] * nc12[4] + ff14[nv4[k]] * nc14[4] + ff14[nv6[k]] * nc16[4] + ff14[nv6[k]] * nc18[4];		f27[k] = f; f277[n f277[n f277[n f277[n f277[n
fisk] = ffis(k) * nc10(5) + ffis(nv1(k)) * nc11(5) + ffi ffis(nv2)k) * nc13(5) + ffi ffis(nv5(k)) * nc15(5) + ffi ffis(nv7(k)) * nc15(5) + ffi	<pre>ff15[nv2[k]] * nc12[5] + ff15[nv4[k]] * nc14[5] + ff15[nv6[k]] * nc16[5] + ff15[nv8[k]] * nc18[5];</pre>		f28[k] = f; f28[n f28[n f28[n f28[n
f16 k] = ff16 k] * nc10 6] + ff1 ff16 nv1 k] * nc11 6] + ff1 ff16 nv3 k] * nc13 6] + ff1 ff16 nv5 k] * nc15 6] + ff1 ff16 nv7 k]] * nc17 6] + ff1	ff16[nv2[k]] * nc12[6] + ff16[nv4[k]] * nc14[6] + ff16[nv6[k]] * nc16[6] + ff16[nv8[k]] * nc18[6];		/* if(k==0) printf(" */ break;
<pre>f17(k) = ff17(k) * nc10(7) + ff1 ff17(nv1(k)) * nc11(7) + ff1 ff17(nv2(k)) * nc15(7) + ff1 ff17(nv5(k)) * nc15(7) + ff1 ff17(nv7(k)) * nc17(7) + ff1</pre>	<pre>ffl7(nv2(k)) * nc12(7) + ffl7(nv4(k)) * nc14(7) + ffl7(nv6(k)) * nc16(7) + ffl7(nv8(k)) * nc18(7);</pre>		0
f18[k] = ff18[k] * nc10[8] + ff18[nv1k]] * nc11[8] + ff1 ff18[nv3[k]] * nc13[8] + ff1 ff18[nv3[k]] * nc13[8] + ff1 ff18[nv7[k]] * nc17[8] + ff1	<pre>ff18[nv2[k]] * nc12[8] + ff18[nv4[k]] * nc14[8] + ff18[nv6[k]] * nc16[8] + ff18[nv8[k]] * nc18[8];</pre>		f12[k] = f ff12[n ff12[n f13[k] = f
ff20[k]; ff21[k] * nc20[1] +			ff13[n f14[k] = f ff14[n
nc21[1] + nc23[1] + nc25[1] + nc27[1] +	ff21[nv4[k]] * nc24[1] + ff21[nv4[k]] * nc24[1] + ff21[nv6[k]] * nc26[1] + ff21[nv8[k]] * nc28[1];		fil4[n f15[k] = f ff15[n ff15[n
f22[k] = ff22[k] * nc20[2] + ff22[nv1[k]] * nc21[2] + ff22[ff22[nv3[k]] * nc23[2] + ff22[ff22[nv5[k]] * nc23[2] + ff22[ff22[nv7[k]] * nc27[2] + ff22[2[nv2[k]] * nc22[2] + 22[nv4[k]] * nc24[2] + 22[nv6[k]] * nc26[2] + 22[nv8[k]] * nc28[2];		고립됩
* nc20[3] nc21[3] + * nc23[3] * nc25[3]	+ ff23[nv2[k]] * nc22[3] + + ff23[nv4[k]] * nc24[3] + + ff23[nv6[k]] * nc26[3] +		f17(k) = f ff17[n ff17[n f18(k) = f

ff23[nv7[k]] * nc27[3] + ff23[nv8[k]] * nc28[3];	
<pre>f24(k) = ff24(k) * nc20[4] + ff24(nv1(k)) * nc21[4] + ff24(nv2(k)) * nc22[4] + ff24(nv3(k)) * nc23[4] + ff24[nv4(k)] * nc24[4] + ff24(nv5(k)) * nc25[4] + ff24(nv6(k)) * nc26[4] + ff24(nv7(k)) * nc27[4] + ff24(nv6(k)) * nc28[4];</pre>	
f25[k] = ff25[k] * nc20[5] + ff25[nv2[k]] * nc21[5] + ff25[nv2[k]] * nc22[5] + ff25[nv3[k]] * nc23[5] + ff25[nv4[k]] * nc24[5] + ff25[nv5[k]] * nc25[5] + ff25[nv6[k]] * nc26[5] + ff25[nv5[k]] * nc27[5] + ff25[nv6[k]] * nc28[5];	
<pre>f26[k] = ff26[k] * nc20[6] + ff26[nv1[k]] * nc21[6] + ff26[nv2[k]] * nc22[6] + ff26[nv3[k]] * nc23[6] + ff26[nv4[k]] * nc24[6] + ff26[nv5[k]] * nc25[6] + ff26[nv6[k]] * nc26[6] + ff26[nv7[k]] * nc27[6] + ff26[nv8[k]] * nc28[6];</pre>	
f27[k] = ff27[k] * nc20[7] + ff27[nv1[k]] * nc21[7] + ff27[nv2[k]] * nc22[7] + ff27[nv4[k]] * nc23[7] + ff27[nv4[k]] * nc24[7] + ff27[nv4[k]] * nc25[7] + ff27[nv6[k]] * nc26[7] + ff27[nv7[k]] * nc27[7] + ff27[nv8[k]] * nc28[7];	
f28[k] = ff28[k] * nc20[8] + ff28[nv1[k]] * nc21[8] + ff28[nv2[k]] * nc22[8] + ff28[nv3[k]] * nc23[8] + ff28[nv4[k]] * nc24[8] + ff28[nv5[k]] * nc25[8] + ff28[nv6[k]] * nc26[8] + ff28[nv7[k]] * nc27[8] + ff28[nv8[k]] * nc28[8];	
/* if(k==0) printf("iter=%d k=%d f1=%e f2=%e\n",iter,k,f10[k],f20[k]); // break; case 1: f10[k] = ff10[k];	
<pre>fil(k) = ffil(k) * ncbot10[1] + ffil[nv1[k]] * ncbot11[1] + ffil[nv2[k]] * ncbot12[1] + ffil[nv3[k]] * ncbot13[1] + ffil[nv5[k]] * ncbot15[1] + ffil[nv6[k]] * ncbot16[1];</pre>	
<pre>f12[k] = ff12[k] * ncbot10[2] + ff12[nv1[k]] * ncbot11[2] +</pre>	
<pre>f13[k] = ff13[k] * ncbot10[3] + ff13[nv1[k]] * ncbot11[3] +</pre>	
<pre>f14(k) = ff14(k) * ncbot10[4] + ff14[nv1(k)] * ncbot11[4] +</pre>	
<pre>f15[k] = ff15[k] * ncbot10[5] + ff15[nv1[k]] * ncbot11[5] + ff15[nv2[k]] * ncbot12[5] + ff15[nv3[k]] * ncbot13[5] + ff15[nv5[k]] * ncbot15[5] + ff15[nv6[k]] * ncbot16[5];</pre>	
<pre>f16(k) = ff16(k) * ncbot10[6] + ff16[nv1(k]] * ncbot11[6] +</pre>	
<pre>f17(k) = ff17(k) * ncbot10[7] + ff17[nv1(k)] * ncbot11[7] + ff17[nv2[k]] * ncbot12[7] + ff17[nv3[k]] * ncbot13[7] + ff17[nv5[k]] * ncbot15[7] + ff17[nv6[k]] * ncbot16[7];</pre>	
f18[k] = ff18[k] * ncbot10[8] + ff18[nv1[k]] * ncbot11[8] +	

0	
5.5	
5.	
ě	
2	
g	
5	
Ξ	
5	
<u>п</u>	
2	
sotone	
თ —	
0	
ĕ	
Printed	
_	

wet9is.c

Jun 20 1999 18:01	Page 13	Jun 20 1999 18:01
<pre>ff18[nv2[k]] * ncbot12[8] + ff18[nv3[k]] * ncbot13[8] + ff18[nv5[k]] * ncbot15[8] + ff18[nv6[k]] * ncbot16[8];</pre>		f18[k] = f1 ff18[nv
f20[k] = ff20[k];		£20[k] = ££
<pre>f21[k] = ff21[k] * ncbot20[1] + ff21[nv1[k]] * ncbot21[1] + ff21[nv2[k]] * ncbot22[1] + ff21[nv3[k]] * ncbot23[1] + ff21[nv5[k]] * ncbot25[1] + ff21[nv6[k]] * ncbot26[1];</pre>	_	[k] = ff21[
<pre>f22[k] = ff22[k] * ncbot20[2] + ff22[nv1[k]] * ncbot21[2] + ff22[nv2[k]] * ncbot22[2] + ff22[nv3[k]] * ncbot23[2] + ff22[nv5[k]] * ncbot25[2] + ff22[nv6[k]] * ncbot26[2];</pre>		
<pre>f23[k] = ff23[k] * ncbot20[3] + ff23[nv1[k]] * ncbot21[3] + ff23[nv2[k]] * ncbot22[3] + ff23[nv3[k]] * ncbot23[3] + ff23[nv5[k]] * ncbot25[3] + ff23[nv6[k]] * ncbot26[3];</pre>		f23[k] = ff ff23[nv
<pre>f24(k) = ff24(k) * ncbot20(4) + ff24(nv1(k)) * ncbot21(4) + ff24(nv2(k)) * ncbot22(4) + ff24(nv3(k)) * ncbot23(4) + ff24(nv5(k)) * ncbot25(4) + ff24(nv6(k)) * ncbot26(4);</pre>		£24[k] = ££ ££24[nv ££24[nv
<pre>f25[k] = ff25[k] * ncbot20[5] + ff25[nv1[k]] * ncbot21[5] + ff25[nv2[k]] * ncbot22[5] + ff25[nv3[k]] * ncbot23[5] + ff25[nv5[k]] * ncbot25[5] + ff25[nv6[k]] * ncbot26[5];</pre>		f25[k] = ff ff25[nv ff25[nv
<pre>f26[k] = ff26[k] * ncbot20[6] + ff26[nv1[k]] * ncbot21[6] + ff26[nv2[k]] * ncbot22[6] + ff26[nv3[k]] * ncbot23[6] + ff26[nv5[k]] * ncbot25[6] + ff26[nv6[k]] * ncbot26[6];</pre>		[26[k] = ff ff26[nv
<pre>f27[k] = ff27[k] * ncbot20[7] + ff27[nv1[k]] * ncbot21[7] + ff27[nv2[k]] * ncbot22[7] + ff27[nv3[k]] * ncbot23[7] + ff27[nv5[k]] * ncbot25[7] + ff27[nv6[k]] * ncbot26[7];</pre>		f27[k] = ff ff27[nv
<pre>f28[k] = ff28[k] * ncbot20[8] + ff28[nv1[k]] * ncbot21[8] + ff28[nv2[k]] * ncbot22[8] + ff28[nv3[k]] * ncbot23[8] + ff28[nv5[k]] * ncbot25[8] + ff28[nv6[k]] * ncbot26[8];</pre>		f28[k] = ff ff28[nv
<pre>break; case 2: f10[k] = ff10[k];</pre>		break; case 3;
<pre>f11[k] = ff11[k] * nctop10[1] + ff11[nv1(k]] * nctop11[1] + ff11[nv3[k]] * nctop13[1] + ff11[nv4[k]] * nctop14[1] + ff11[nv7[k]] * nctop17[1] + ff11[nv8[k]] * nctop18[1];</pre>		# E
<pre>f12[k] = ff12[k] * nctop10[2] + ff12[nv1[k]] * nctop11[2] + ff12[nv3[k]] * nctop13[2] + ff12[nv4[k]] * nctop14[2] + ff12[nv7[k]] * nctop17[2] + ff12[nv8[k]] * nctop18[2];</pre>		f12[k] = f1 ff12[m
<pre>f13(k] = ff13(k) * nctop10(3) + ff13(nv1(k)) * nctop11(3) + ff13(nv3(k)) * nctop13(3) + ff13(nv4(k)) * nctop14(3) + ff13(nv7(k)) * nctop17(3) + ff13(nv8(k)) * nctop18(3);</pre>		f13(k) = ff f13(k) = ff
<pre>f14(k) = ff14(k) * nctop10(4) + ff14(nv1(k)) * nctop11(4) +</pre>		f14[k] = f1 f14[h] = f1 f14[n]
<pre>f15[k] = ff15[k] * nctop10[5] + ff15[nv1[k]] * nctop11[5] + ff15[nv3[k]] * nctop13[5] + ff15[nv4[k]] * nctop14[5] + ff15[nv7[k]] * nctop17[5] + ff15[nv8[k]] * nctop18[5];</pre>		f15[k] = f1 ff15[n ff15]n
* * *		£16[k] = £1 £116[n\ ££16[n\
<pre>f17[k] = ff17[k] * nctop10[7] + ff17[nv1[k]] * nctop11[7] + ff17[nv2[k]] * nctop13[7] + ff17[nv4[k]] * nctop14[7] + ff17[nv7[k]] * nctop17[7] + ff17[nv8[k]] * nctop18[7];</pre>		£17[k] = £1 £17[n

<pre>f18[k] = ff18[k] * nctop10[8] + ff18[nv1[k]] * nctop11[8] + ff18[nv3[k]] * nctop13[8] + ff18[nv4[k]] * nctop14[8] + ff18[nv7[k]] * nctop17[8] + ff18[nv8[k]] * nctop18[8];</pre>
<pre>f20(k] = ff20[k]; f21[k] = ff21[k] * nctop20[1] + ff21[nv1[k]] * nctop21[1] +</pre>
<pre>[nv7[k]] * nctop27[1] + ff21[nv8[k]] * nctop28[1]; ff22[k] * nctop20[2] + ff22[nv1[k]] * nctop21[2] [nv3[k]] * nctop23[2] + ff22[nv4[k]] * nctop24[2] [nv7[k]] * nctop27[2] + ff22[nv8[k]] * nctop28[2];</pre>
ff23(k) * nctop20[3] + ff23[nv1(k]] * nctop21[3] nv3(k)] * nctop23[3] + ff23[nv4(k]] * nctop24[3] inv7(k)] * nctop27[3] + ff23[nv8(k]] * nctop28[3]
<pre>f24[k] = ff24[k] * nctop20[4] + ff24[nv1[k]] * nctop21[4] + ff24[nv3[k]] * nctop23[4] + ff24[nv4[k]] * nctop24[4] + ff24[nv7[k]] * nctop27[4] + ff24[nv8[k]] * nctop28[4];</pre>
<pre>f25[k] = ff25[k] * nctop20[5] + ff25[nv1[k]] * nctop21[5] + ff25[nv3[k]] * nctop23[5] + ff25[nv4[k]] * nctop24[5] + ff25[nv7[k]] * nctop27[5] + ff25[nv8[k]] * nctop28[5];</pre>
<pre>f26(k) = ff26(k) * nctop20(6] + ff26[nv1(k)] * nctop21(6] + ff26[nv3(k)] * nctop23(6] + ff26[nv4(k)] * nctop24(6] + ff26[nv7(k)] * nctop27(6] + ff26[nv8(k)] * nctop28(6);</pre>
<pre>f27(k] = ff27(k] * nctop20[7] + ff27[nv1 k]] * nctop21[7] + ff27[nv3[k]] * nctop23[7] + ff27[nv4[k]] * nctop24[7] + ff27[nv7[k]] * nctop27[7] + ff27[nv8[k]] * nctop28[7];</pre>
<pre>f28[k] = ff28[k] * nctop20[8] + ff28[nv1[k]] * nctop21[8] + ff28[nv3[k]] * nctop23[8] + ff28[nv4[k]] * nctop24[8] + ff28[nv7[k]] * nctop27[8] + ff28[nv8[k]] * nctop28[8];</pre>
<pre>break; case 3: fl0[k] = ffl0[k];</pre>
<pre>fil(k) = ffil(k) * ncleft10[1] + ffil(nv1[k]) * ncleft11[1] + ffil[nv5[k]] * ncleft15[1] + ffil[nv2[k]] * ncleft12[1] + ffil[nv4[k]] * ncleft14[1] + ffil[nv8[k]] * ncleft18[1];</pre>
<pre>f12[k] = ff12[k] * ncleft10[2] + ff12[nv1[k]] * ncleft11[2] + ff12[nv5[k]] * ncleft15[2] + ff12[nv2[k]] * ncleft12[2] + ff12[nv4[k]] * ncleft14[2] + ff12[nv8[k]] * ncleft18[2];</pre>
<pre>f13(k) = ff13(k) * ncleft10(3) + ff13(nv1(k)) * ncleft11(3) + ff13[nv5(k]) * ncleft15(3) + ff13[nv2(k)] * ncleft12(3) + ff13[nv4(k)] * ncleft14(3) + ff13[nv8(k)] * ncleft18(3);</pre>
<pre>f14[k] = ff14[k] * ncleft10[4] + ff14[nv1[k]] * ncleft11[4] + ff14[nv5[k]] * ncleft15[4] + ff14[nv2[k]] * ncleft12[4] + ff14[nv4[k]] * ncleft14[4] + ff14[nv8[k]] * ncleft18[4];</pre>
<pre>f15[k] = ff15[k] * ncleft10[5] + ff15[nv1[k]] * ncleft11[5] + ff15[nv5[k]] * ncleft15[5] + ff15[nv2[k]] * ncleft12[5] + ff15[nv4[k]] * ncleft14[5] + ff15[nv8[k]] * ncleft18[5];</pre>
<pre>f16(k) = ff16(k) * ncleft10[6] + ff16[nv1(k)] * ncleft11[6] + ff16[nv5[k]] * ncleft15[6] + ff16[nv2[k]] * ncleft12[6] + ff16[nv4[k]] * ncleft14[6] + ff16[nv8[k]] * ncleft18[6];</pre>
<pre>f17(k] = ff17(k] * ncleft10[7] + ff17[nv1(k]] * ncleft11[7] + ff17[nv5[k]] * ncleft15[7] + ff17[nv2[k]] * ncleft12[7] + ff17[nv4[k]] * ncleft14[7] + ff17[nv8[k]] * ncleft18[7];</pre>

_
2
Ŧ
ပ္
Ĕ
က္က
ğ
듬
≡
늗
₽
ea
5
ō
8
Ō.
eg
Ξ
7

Jun 20 1999 18:01 Wet9is.c	Page 15	Jun 20 1999 18:01
101 102 103		ff17[nv
<pre>118 K] = II18 M. M.</pre>		f18[k] = ff ff18[nv ff18[nv
f20[k] = ff20[k];		£201163 ==
<pre>f21(k) = ff21(k) * ncleft20[1] + ff21(nv1[k]) * ncleft21[1] + ff21[nv5[k]] * ncleft25[1] + ff21[nv2[k]] * ncleft22[1] + ff21[nv4[k]] * ncleft24[1] + ff21[nv8[k]] * ncleft28[1];</pre>	-	CA C
<pre>f22(k) = ff22(k) * ncleft20(2) + ff22(nv1(k)) * ncleft21(2) +</pre>		f22[k] = ff ff22[nV ff22[nV
<pre>f23[k] = ff23[k] * ncleft20[3] + ff23[nv1[k]] * ncleft21[3] +</pre>		f23 k] = ff ff23 nv ff23 nv
<pre>£24[k] = f£24[k] * ncleft20[4] + f£24[nv1[k]] * ncleft21[4] +</pre>		[24 k] = EE [E24 nv ff24 nv ff24 nv
<pre>f25[k] = ff25[k] * noleft20[5] + ff25[nv1[k]] * noleft21[5] +</pre>	- N#34	######################################
<pre>f26(k) = ff26(k) * ncleft20(6] + ff26(nv1(k)) * ncleft21[6] + ff26(nv5(k)) * ncleft25[6] + ff26(nv2(k)) * ncleft22[6] + ff26(nv4(k)) * ncleft24[6] + ff26(nv8(k)) * ncleft28[6];</pre>		f26[k] = ff f26[nv ff26[nv
<pre>f27[k] = ff27[k] * ncleft20[7] + ff27[nv1[k]] * ncleft21[7] + ff27[nv5[k]] * ncleft25[7] + ff27[nv2[k]] * ncleft22[7] + ff27[nv4[k]] * ncleft24[7] + ff27[nv8[k]] * ncleft28[7];</pre>		f27[k] = ff f27[k] = ff
<pre>f28[k] = ff28[k] * ncleft20[8] + ff28[nv1[k]] * ncleft21[8] + ff28[nv5[k]] * ncleft25[8] + ff28[nv2[k]] * ncleft22[8] + ff28[nv4[k]] * ncleft24[8] + ff28[nv8[k]] * ncleft28[8];</pre>		[28[k] = [58] [58]
<pre>break; case 4: f10[k] = ff10[k];</pre>		break;
<pre>fil(k] = ffll(k] * ncrightl0[1] + ffll[nv3[k]] * ncrightl3[1] + ffll[nv6[k]] * ncrightl6[1] + ffll[nv2[k]] * ncrightl2[1] + ffll[nv4[k]] * ncrightl4[1] + ffll[nv7[k]] * ncrightl7[1];</pre>		} (void iup (void)
<pre>f12[k] = ff12[k] * ncright10[2] + ff12[nv3[k]] * ncright13[2] + ff12[nv6[k]] * ncright16[2] + ff12[nv2[k]] * ncright12[2] + ff12[nv4[k]] * ncright14[2] + ff12[nv7[k]] * ncright17[2];</pre>		int k; for (k=0; k <nnodes_a {</nnodes_a
<pre>f13[k] = ff13[k] * ncright10[3] + ff13[nv3[k]] * ncright13[3] + ff13[nv6[k]] * ncright16[3] + ff13[nv2[k]] * ncright12[3] + ff13[nv4[k]] * ncright14[3] + ff13[nv7[k]] * ncright17[3];</pre>		
<pre>f14[k] = ff14[k] * ncright10[4] + ff14[nv3[k]] * ncright13[4] + ff14[nv6[k]] * ncright16[4] + ff14[nv2[k]] * ncright12[4] + ff14[nv4[k]] * ncright14[4] + ff14[nv7[k]] * ncright17[4];</pre>		ff11[nv3] ff11[nv3]
<pre>f15[k] = ff15[k] * ncright10[5] + ff15[nv3[k]] * ncright13[5] + ff15[nv6[k]] * ncright16[5] + ff15[nv2[k]] * ncright12[5] + ff15[nv4[k]] * ncright14[5] + ff15[nv7[k]] * ncright17[5];</pre>		
<pre>f16[k] = ff16[k] * ncright10[6] + ff16[nv3[k]] * ncright13[6] +</pre>		f13[k] = ff13[k] ff13[k] = ff13[k]
<pre>f17[k] = ff17[k] * ncright10[7] + ff17[nv3[k]] * ncright13[7] +</pre>		ff13[nv3] ff13[nv3]

	££17[nv4[k]]	* ncright14[7]	+	ff17[nv7[k]]	*	ncright17[7]	11;
f1	8[k] = ff18[k] ff18[nv6[k]] ff18[nv4[k]]	* ncright10[8] * ncright16[8] * ncright14[8]	+++	ff18[nv3[k]] ff18[nv2[k]] ff18[nv7[k]]	* * *	ncright13[8] ncright12[8] ncright17[8]	++
44	f20[k] = ff20[k];						
ų.	f21[k] = ff21[k] ff21[nv6[k]] ff21[nv4[k]]	* ncright20[1] * ncright26[1] * ncright24[1]	+++	ff21[nv3[k]] ff21[nv2[k]] ff21[nv7[k]]	* * *	ncright23[1] ncright22[1] ncright27[1];	++
4 1	f22[k] = ff22[k] ff22[nv6[k]] ff22[nv4[k]]	* ncright20[2] * ncright26[2] * ncright24[2]	+++	ff22[nv3[k]] ff22[nv2[k]] ff22[nv7[k]]	* * *	ncright23[2] ncright22[2] ncright27[2];	++ :
44	f23[k] = ff23[k] ff23[nv6[k]] ff23[nv4[k]]	* ncright20[3] * ncright26[3] * ncright24[3]	+ + +	ff23[nv3[k]] ff23[nv2[k]] ff23[nv7[k]]	* * *	ncright23[3 ncright22[3 ncright27[3	++.
44	f24[k] = ff24[k] ff24[nv6[k]] ff24[nv4[k]]	* ncright20[4] * ncright26[4] * ncright24[4]	+++	ff24[nv3[k]] ff24[nv2[k]] ff24[nv7[k]]	* * *	ncright23[4] ncright22[4] ncright27[4];	++.
44	f25[k] = ff25[k] ff25[nv6[k]] ff25[nv4[k]]	* ncright20[5] * ncright26[5] * ncright24[5]	+++	ff25[nv3[k]] ff25[nv2[k]] ff25[nv7[k]]	* * *	ncright23[5] ncright22[5] ncright27[5]	++:
44	f26[k] = ff26[k] ff26[nv6[k]] ff26[nv4[k]]	* ncright20[6] * ncright26[6] * ncright24[6]	+ + +	ff26[nv3[k]] ff26[nv2[k]] ff26[nv7[k]]	* * *	ncright23[6] ncright22[6] ncright27[6];	++
44	f27[k] = ff27[k] ff27[nv6[k]] ff27[nv4[k]]	* ncright20[7] * ncright26[7] * ncright24[7]	+++	ff27[nv3[k]] ff27[nv2[k]] ff27[nv7[k]]	* * *	ncright23[7] ncright22[7] ncright27[7]	++
44	f28[k] = ff28[k] ff28[nv6[k]] ff28[nv4[k]]	* ncright20[8] * ncright26[8] * ncright24[8]	+++	ff28[nv3[k]] ff28[nv2[k]] ff28[nv7[k]]	* * *	ncright23[8] ncright22[8] ncright27[8]	++.
Q (break;						
void iup (void)	id)						
	<pre>int k; for(k=0; k<nnodes_all; k+<="" pre=""></nnodes_all;></pre>	k++)					
f10[k]	:] = ff10[k];						
#111, ff1 ff1 ff1 ff1 ff1	<pre>fil(k] = ffil(k] * nui5[i] + ffil(nv3[k]) * nui5[i] + ffil(nv3[nv3[k]) * nui6[i] ffil(nv4[k]) * nui1[i] + ffil(nv7[k]) * nui0[i] + ffil(nv3[nv7[k]) * nui1[i] + ffil(nv4[nv4[k]) * nui ffil(nv3[nv4[k])] * nui4[i] + ffil(nv7[nv7[k])]</pre>	115[1] + ff11[nv] [1] + ff11[nv] [1] + ff11[nv] nu13[1] + ff1]	3[nv3 7[k]] 11[nv + ff	[k]]] * nul * nul0[1] 4[nv4[k]]] 11[nv7[nv7[]	6[1] + * nu1 k]]]	[1] + nu18[1] +]]] * nu17[1]	96.
112[k] ff1] ff1] ff1]	[2[k] = ffl2[k] * nul5[2] + ffl2[nv3[k]]] * nul5[2] + ffl2[nv3[nv3[k]]] * nul6[2] - ffl2[nv3[k]] * nul1[2] + ffl2[nv7[k]] * nul0[2] + ffl2[nv3[nv7[k]]] * nul3[2] + ffl2[nv4[nv4[k]]] * nul ffl2[nv3[nv4[nv4[k]]]] * nul4[2] + ffl2[nv7[nv7[k]]]]	15[2] + ff12[nv] [2] + ff12[nv] [2] + ff12[nv] nu13[2] + ff]]]] * nu14[2]	3[nv3 7[k]] 12[nv + ff	[k]]] * nu1 * nu10[2] 4[nv4[k]]] 12[nv7[nv7[]	6[2] + * nu1 k]]]	+ 8[2] + * nul7[2];
113.00 11	f13[k] = ff13[k] * nu15[3] + ff13[nv3[k]] * nu16[3] + ff13[nv3[k]]] * nu16[3] + ff13[nv4[k]] * nu11[3] + ff13[nv7[k]] * nu10[3] + ff13[nv3[nv7[k]] * nu11[3] + ff13[nv4[nv4[k]]] * nu18[3] ff13[nv3[nv4[nv4[k]]]] * nu14[3] + ff13[nv7[nv7[k]]] * nu18[3]	115[3] + 2[3] + ££13[nv; [[3] + ££13[nv; 10] + ££13[nv; 10]] + ££13[nv;	3[nv3 7[k]] 13[nv + ff	[k]]] * nu1 * nu10[3] 4[nv4[k]]] 13[nv7[nv7]	6[3] + * nu1 k]]]	+ .8[3] + * nul7[3];	45

	-1
-	1
Ö	1
#	1
±	1
╼	1
mec.u	1
ပ	1
Ø.	1
-	1
=	1
_=	1
- 62	1
0	1
ത	1
_	١
	J
	Į
€	1
400	Į
_	ı
⊑	ı
0	1
_	ų
4-	1
- crt	1
- 22	1
ofone	1
	1
0	1
₽	1
0	Ц
ū	1
	Н
\sim	1
0	1
_	ł
-0	1
Ø	1
==	1
inte	1
-∈	1
~	ì
-	Į
	1
	1
	1
	1
	١
	1
	1
	1
	1

	Jun 20 1999 18:01 Wet9Is.c	Page 17	Jun 20 1999 18:01
### ### ### ### #### #### #### ########	= ff14[k] * nu15[4] + [nv3[k]] * nu12[4] + ff14[nv3[nv3[k]]] * nu16[4] + [nv4[k]] * nu11[4] + ff14[nv7[k]] * nu10[4] + [nv3[nv7[k]]] * nu13[4] + ff14[nv4[nv4[k]]] * nu18[4] + [nv3[nv4[k]]]] * nu14[4] + ff14[nv7[kv]] * nu17[4]		
y, ', ', ', ', ', ', ', ', ', ', ', ', ',	= ff15[k] * nu15[5] + [nv3[k]] * nu12[5] + ff15[nv3[nv3[k]]] * nu16[5] + [nv4[k]] * nu11[5] + ff15[nv7[k]] * nu10[5] + [nv3[nv7[k]]] * nu13[5] + ff15[nv4[nv4[k]]] * nu18[5] + [nv3[nv4[nv4[k]]]] * nu14[5] + ff15[nv4[nv7[k]]] * nu17[5]		2222
void int for (for () () () () () () () () () (<pre>fig(k) = ffl6(k) * nul2(6) + ffl6(nv3(k)) * nul2(6) + ffl6(nv3(k))) * nul6(6) + ffl6(nv4(k)) * nul1(6) + ffl6(nv7(k)) * nul0(6) + ffl6(nv3(nv7(k))) * nul3(6) + ffl6(nv4(nv4(k))) * nul8(6) + ffl6(nv3(nv4(nv4(k))) * nul4(6) + ffl6(nv7(nv7(k))) * nul7(6);</pre>		_
	<pre>f17(k) = ff17(k) * nu15(7) + ff17(nv3(k)] * nu12(7) + ff17(nv3(k)]) * nu16(7) + ff17(nv3(k)] * nu11(7) + ff17(nv7(k)] * nu10(7) + ff17(nv3(nv7(k))) * nu13(7) + ff17(nv4(nv4(k))) * nu18(7) + ff17(nv3(nv4(k))) * nu14(7) + ff17(nv7(nv7(k))) * nu17(7);</pre>		void ifd(void) { int k; for(k=0; k <nnodes.< td=""></nnodes.<>
ff13 ff14 ff15 ff16	f18[k] = ff18[k] * nu12[8] + ff18[nv3[k]] * nu16[8] + ff18[nv3[k]] * nu12[8] + ff18[nv7[k]] * nu16[8] + ff18[nv4[k]] * nu11[8] + ff18[nv7[k]] * nu16[8] + ff18[nv3[nv4[k]]] * nu13[8] + ff18[nv4[nv4[k]]] * nu18[8] + ff18[nv3[nv4[nv4[k]]] * nu14[8] + ff18[nv7[nv7[k]]] * nu17[8];		(case 0; ff10[k] = ff11[k] = ff11[k]
££131 ££14 ££15 ££15	f20[k] = ff20[k];		71117 71117 71117
ff13 ff14 ff15 ff15	<pre>f21[k] = ff21[k] * nu35[1] + ff21[nv3[h] * nu25[1] * nu22[1] * nu22[1] * nu22[1] + ff21[nv4[k]] * nu21[1] + ff21[nv7[k]] * nu20[1] + ff21[nv4[k]] * nu21[1] + ff21[nv4[k]] * nu28[1] + ff21[nv3[nv7[k]]] * nu23[1] + ff21[nv4[kv4[k]]] * nu28[1] + ff21[nv3[nv4[k]]] * nu24[1] + ff21[nv7[nv7[k]]] * nu27[1];</pre>		ff12[k] = ff12[k] = f12[k] = f
ff15 ff15 ff17	f22[k] = ff22[k] * nu25[2] + ff22[nv3[k]] * nu22[2] + ff22[nv3[nv3[k]]] * nu26[2] + ff22[nv4[k]] * nu21[2] + ff22[nv7[k]] * nu20[2] + ff22[nv3[nv7[k]]] * nu23[2] + ff22[nv4[nv4[k]]] * nu28[2] + ff22[nv3[nv7[k]]] * nu24[2] + ff22[nv7[nv4[k]]] * nu27[2];		ff13(K) = ff13(K) = f13(F) = f
ff15[f23[k] = ff23[k] * nu22[3] + ff23[nv3[k]] * nu22[3] + ff23[nv3[nv3[k]]] * nu26[3] + ff23[nv4[k]] * nu21[3] + ff23[nv7[k]] * nu20[3] + ff23[nv3[nv7[k]]] * nu23[3] + ff23[nv4[nv4[k]]] * nu28[3] + ff23[nv3[nv4[nv4[k]]] * nu24[3] + ff23[nv4[nv4[k]]] * nu28[3] +		ff14[K] ff14[K] ff14[F] ff14[F
	f24[k] = ff24[k] * nu25[4] + ff24[nv3[k]]] * nu22[4] + ff24[nv3[nv3[k]]] * nu26[4] + ff24[nv4[k]] * nu21[4] + ff24[nv7[k]] * nu20[4] + ff24[nv3[nv7[k]]] * nu23[4] + ff24[nv4[nv4[k]]] * nu28[4] + ff24[nv3[nv4[k]]] * nu24[4] + ff24[nv4[nv4[k]]] * nu28[4] +		ff15(1) ff15(1) f15(1) f15(1)
££17[f25[k] = ff25[k] * nu25[5] + ff25[nv3[k]] * nu26[5] + ff25[nv3[nv3[k]]] * nu26[5] + ff25[nv4[k]] * nu21[5] + ff25[nv7[k]] * nu20[5] + ff25[nv3[nv7[k]]] * nu23[5] + ff25[nv4[nv4[k]]] * nu28[5] + ff25[nv3[nv4[k]]] * nu24[5] + ff25[nv4[nv4[k]]] * nu28[5] +		1511 (1)9113 (1)911
	f26[k] = ff26[k] * nu25[6] + ff26[nv3[k]]] * nu26[6] + ff26[nv3[k]] * nu20[6] + ff26[nv3[nv7[k]] * nu20[6] + ff26[nv4[k]] * nu21[6] + ff26[nv4[kv4[k]] * nu28[6] + ff26[nv3[nv7[k]]] * nu23[6] + ff26[nv4[nv4[k]]] * nu28[6] + ff26[nv3[nv4[nv4[k]]] * nu24[6] + ff26[nv7[nv7[k]]] * nu27[6];		£16[1] ££17[k] = £17[1] £17[1]

<pre>£27[k] = ££27[k] * nu25[7] + ££27[nv3[k]] * nu26[7] + ££27[nv3[k]] * nu22[7] + ££27[nv3[k]] * nu26[7] + ££27[nv4[k]] * nu21[7] + ££27[nv7[k]] * nu20[7] + ££27[nv3[nv7[k]]] * nu23[7] + ££27[nv4[nv4[k]]] * nu28[7] + ££27[nv3[nv4[kv4[k]]] * nu24[7] + ££27[nv7[nv7[k]]] * nu27[7];</pre>	8[k] = ff28[k] * nu25[8] + ff28[nv3[k]] * nu22[8] + ff28[nv3[nv3[k]]] * nu26[8] + ff28[nv4[k]] * nu21[8] + ff28[nv7[k]] * nu20[8] + ff28[nv3[nv7[k]]] * nu23[8] + ff28[nv4[nv4[k]]] * nu28[8] + ff28[nv3[nv4[nv4[k]]]] * nu24[8] + ff28[nv7[nv7[k]]] * nu27[8];	fl=%e f2=%e\n",iter,k,fl0[k],f20[k]);					nc10[1] + f11[nv2[k]] * nc12[1] + nc13[1] + f11[nv4[k]] * nc14[1] + nc15[1] + f11[nv6[k]] * nc16[1] + nc15[1] + f11[nv8[k]] * nc18[1];	nc10[2] + f12[nv2[k]] * nc12[2] + nc13[2] + f12[nv4[k]] * nc14[2] + nc15[2] + f12[nv6[k]] * nc14[2] + nc17[2] + f12[nv8[k]] * nc18[2];	nc10[3] + fi3[nv2[k]) * nc12[3] + nc13[3] + fi3[nv4[k]] * nc14[3] + fi3[nv4[k]] * nc14[3] + nc15[3] + fi3[nv8[k]] * nc16[3] + nc17[3] + fi3[nv8[k]] * nc18[3];	nci0[4] + ncil[4] + fl4[nv2[k]] * nci2[4] + nci3[4] + fl4[nv4[k]] * nci4[4] + nci5[4] + fl4[nv6[k]] * nci6[4] + nci5[4] + fl4[nv8[k]] * nci8[4];	nc10[5] + f15[nv2[k]] * nc12[5] + nc13[5] + f15[nv4[k]] * nc14[5] + f15[nv4[k]] * nc14[5] + nc15[5] + f15[nv8[k]] * nc18[5] + f15[nv8[k]] * nc18[5];	nc10[6] + f16[nv2[k]] * nc12[6] + nc13[6] + f16[nv4[k]] * nc14[6] + nc15[6] + f16[nv8[k]] * nc14[6] + nc15[6] + f16[nv8[k]] * nc18[6];	ncl0[7] + f17[nv2[k]] * ncl2[7] + ncl3[7] + f17[nv4[k]] * ncl4[7] + ncl5[7] + f17[nv6[k]] * ncl6[7] +
f27[k] = ff27[k] * nu25[ff27[nv3[k]] * nu22[7] ff27[nv4[k]] * nu21[7] if27[nv4[k]]] * nu21[7] if27[nv3[nv7[k]] * nu ff27[nv3[nv4[k]]]] * nu	f28[k] = ff28[k] * nu25[ff28[nv3[k]] * nu22[8] ff28[nv4[k]] * nu21[8] iff28[nv4[k]] * nu2[6] iff28[nv3[nv4[k]]] * nu2[6]	/* if(k==0) printf("iter=%d k=%d	void ifd(void)	int k; for(k=0; k <nnodes_all; k++)<="" td=""><td>switch(boundary_mode[k])</td><td>case 0: ff10[k] = f10[k];</td><td>ffll[k] = fll[k] * n fll[nv1[k]] * n fll[nv3[k]] * n fll[nv3[k]] * n fll[nv7[k]] * n</td><td>ff12[k] = f12[k] * n f12[nv1[k]] * n f12[nv3[k]] * n f12[nv5[k]] * n f12[nv7[k]] * n</td><td>ff13[k] = f13[k] * n f13[nv1[k]] * n f13[nv3[k]] * n f13[nv5[k]] * n f13[nv7[k]] * n</td><td>ff14[k] = f14[k] * r f14[nv1[k]] * r f14[nv3[k]] * r f14[nv3[k]] * r f14[nv5[k]] * r</td><td>ff15[k] = f15[k] * r f15[nv3[k]] * r f15[nv3[k]] * r f15[nv3[k]] * r f15[nv5[k]] * r</td><td>ff16[k] = f16[k] * r f16[nv3[k]] * r f16[nv3[k]] * r f16[nv3[k]] * r f16[nv5[k]] * r</td><td>ff17[k] = f17[k] * 7 f17[nv1[k]] * 7 f17[nv3[k]] * 7 f17[nv5[k]] * 1</td></nnodes_all;>	switch(boundary_mode[k])	case 0: ff10[k] = f10[k];	ffll[k] = fll[k] * n fll[nv1[k]] * n fll[nv3[k]] * n fll[nv3[k]] * n fll[nv7[k]] * n	ff12[k] = f12[k] * n f12[nv1[k]] * n f12[nv3[k]] * n f12[nv5[k]] * n f12[nv7[k]] * n	ff13[k] = f13[k] * n f13[nv1[k]] * n f13[nv3[k]] * n f13[nv5[k]] * n f13[nv7[k]] * n	ff14[k] = f14[k] * r f14[nv1[k]] * r f14[nv3[k]] * r f14[nv3[k]] * r f14[nv5[k]] * r	ff15[k] = f15[k] * r f15[nv3[k]] * r f15[nv3[k]] * r f15[nv3[k]] * r f15[nv5[k]] * r	ff16[k] = f16[k] * r f16[nv3[k]] * r f16[nv3[k]] * r f16[nv3[k]] * r f16[nv5[k]] * r	ff17[k] = f17[k] * 7 f17[nv1[k]] * 7 f17[nv3[k]] * 7 f17[nv5[k]] * 1

į
ı
١

Jun 20 1999 18:01	Wet91S.C	rage 18
£17[nv7[k]] * nc17	[7] + f17[nv8[k]] * nc18[7];	
ff18[k] = f18[k] * nc10[f18[nv1[k]] * nc11[f18[nv3[k]] * nc13[f18[nv5[k]] * nc13[f18[nv5[k]] * nc17[[8] + f18[nv2[k]] * nc12[8] + [8] + f18[nv4[k]] * nc14[8] + [8] + f18[nv6[k]] * nc16[8] + [8] + f18[nv8[k]] * nc18[8];	
ff20[k] = f20[k];		
ff21[k] = f21[k] * nc20[f21[nv1[k]] * nc31[f21[nv3[k]] * nc23[f21[nv5[k]] * nc27[f21[nv7[k]] * nc27[[1] + f21[nv2[k]] * nc22[1] + f21[nv4[k]] * nc24[1] + f21[nv4[k]] * nc24[1] + f21[nv6[k]] * nc26[1] + f21[nv8[k]] * nc28[1];	
ff22[k] = f22[k] * nc20[f22[nv1[k]] * nc31[f22[nv3[k]] * nc31[f22[nv5[k]] * nc25[f22[nv7[k]] * nc27[[2] + f22[nv2[k]] * nc22[2] + [2] + f22[nv4[k]] * nc24[2] + [2] + f22[nv4[k]] * nc26[2] + [2] + f22[nv6[k]] * nc26[2];	
ff23[k] = f23[k] * nc20[f23[nv1[k]] * nc21[f23[nv3[k]] * nc23[f23[nv5[k]] * nc27[f23[nv7[k]] * nc27[[3] + f23[nv2[k]] * nc22[3] + [3] + f23[nv4[k]] * nc24[3] + [3] + f23[nv6[k]] * nc26[3] + [3] + f23[nv6[k]] * nc26[3];	
ff24[k] = f24[k] * nc20[f24[nv1[k]] * nc21[f24[nv3[k]] * nc23[f24[nv5[k]] * nc27[f24[nv7[k]] * nc27[[4] + f24[nv2[k]] * nc22[4] + f4 + f24[nv4[k]] * nc24[4] + f4 + f24[nv6[k]] * nc26[4] + f24[nv8[k]] * nc28[4];	
ff25[k] = f25[k] * nc20[5] f25[nv1[k]] * nc21[5] f25[nv3[k]] * nc21[5] f25[nv5[k]] * nc25[5] f25[nv7[k]] * nc25[5]	[[5] + f25[nv2[k]] * nc22[5] + [5] + f25[nv4[k]] * nc24[5] + [15] + f25[nv6[k]] * nc26[5] + [15] + f25[nv6[k]] * nc26[5] + [15] + f25[nv6[k]] * nc28[5];	
ff26[k] = f26[k] * nc20[6] f26[nv1[k]] * nc21[6] f26[nv3[k]] * nc23[6] f26[nv5[k]] * nc25[6] f26[nv7[k]] * nc25[6]	[6] + f26[nv2[k]] * nc22[6] + [6] + f26[nv4[k]] * nc24[6] + [6] + [5] + [6] +	
ff27[k] = f27[k] * nc20[7] f27[nv1[k]] * nc21[7] f27[nv3[k]] * nc23[7] f27[nv3[k]] * nc25[7] f27[nv7[k]] * nc25[7]		
ff28[k] = f28[k] * nc20[8 f28[nv1[k]] * nc21[8 f28[nv3[k]] * nc23[8 f28[nv5[k]] * nc25[8 break;	[8] + £28[nv2[k]] * nc22[8] + [8] + £28[nv4[k]] * nc24[8] + [8] + £28[nv6[k]] * nc26[8] + [8] + £28[nv8[k]] * nc28[8];	
case 1: ff10[k] = f10[k];		
<pre>ffll(k] = fll(k) * ncbo fll(nv2[k]] * ncbo fll(nv5[k]) * ncbo</pre>	<pre>ncbot10[1] + f11[nv1[k]] * ncbot11[1] + ncbot12[1] + f11[nv3[k]] * ncbot13[1] + ncbot15[1] + f11[nv6[k]] * ncbot16[1];</pre>	
<pre>ff12[k] = f12[k] * ncbo f12[nv2[k]] * ncbo f12[nv5[k]] * ncbo</pre>	ncbot10[2] + f12[nv1[k]] * ncbot11[2] + ncbot12[2] + f12[nv3[k]] * ncbot13[2] + ncbot15[2] + f12[nv6[k]] * ncbot16[2];	

1,000 1000 1000	A sight	OC OCC
3	SEIGH	r age zo
ff13[k] = f13[k] *	ncbot10[3] + f13[nv1[k]] *	ncbot11[3] +
f13[nv2[k]] *	ncbot12[3] + f13[nv3[k]] *	ncbot13[3] +
f13[nv5[k]] *	ncbot15[3] + f13[nv6[k]] *	ncbot16[3];
ff14[k] = f14[k] *	ncbot10[4] + £14[nv1[k]] * ncbot11[4]	11[4] +
f14[nv2[k]] *	ncbot12[4] + £14[nv3[k]] * ncbot13[4]	13[4] +
f14[nv5[k]] *	ncbot15[4] + £14[nv6[k]] * ncbot16[4]	16[4];
<pre>ffl5[k] = f15[k] * f15[nv2[k]] * f15[nv5[k]] *</pre>	ncbotl0[5] + f15[nv1[k]] * ncbotl1[5] ncbotl2[5] + f15[nv3[k]] * ncbotl3[5] ncbotl5[5] + f15[nv6[k]] * ncbotl6[5];	11[5] + 13[5] + 16[5];
<pre>ff16[k] = f16[k] * f16[nv2[k]] * f16[nv5[k]] *</pre>	ncbot10[6] + f16[nv1[k]] * ncbot11[6] ncbot12[6] + f16[nv3[k]] * ncbot13[6] ncbot15[6] + f16[nv6[k]] * ncbot16[6];	11[6] + 13[6] + 16[6];
ff17[k] = f17[k] *	ncbot10[7] + f17[nv1[k]] * ncbot11[7]	11[7] +
f17[nv2[k]] *	ncbot12[7] + f17[nv3[k]] * ncbot13[7]	13[7] +
f17[nv5[k]] *	ncbot15[7] + f17[nv6[k]] * ncbot16[7];	16[7];
<pre>ffl8[k] = f18[k] * f18[nv2[k]] * f18[nv5[k]] *</pre>	<pre>r ncbot10[8] + f18[nv1[k]] * ncbot11[8] r ncbot12[8] + f18[nv3[k]] * ncbot13[8] r ncbot15[8] + f18[nv6[k]] * ncbot16[8];</pre>	11[8] + 13[8] + 16[8];
ff20[k] = f20[k];		
ff21[k] = f21[k] *	* ncbot20[1] + f21[nv][k]] * ncbot21[1]	21(1) +
f21[nv2[k]] *	* ncbot22[1] + f21[nv3[k]] * ncbot23[1]	23(1) +
f21[nv5[k]] *	* ncbot25[1] + f21[nv6[k]] * ncbot26[1];	26(1);
ff22[k] = f22[k] * f22[nv2[k]] * f22[nv5[k]] *	<pre>ncbot20[2] + f22[nv1[k]] * ncbot21[2] ncbot22[2] + f22[nv3[k]] * ncbot23[2] ncbot25[2] + f22[nv6[k]] * ncbot26[2];</pre>	21[2] + 23[2] + 26[2];
ff23[k] = f23[k] *	* ncbot20[3] + f23[nv1[k]] * ncbot21[3]	21[3] +
f23[nv2[k]] *	* ncbot22[3] + f23[nv3[k]] * ncbot23[3]	23[3] +
f23[nv5[k]] *	* ncbot25[3] + f23[nv6[k]] * ncbot26[3];	26[3];
ff24[k] = f24[k] *	ncbot20[4] + f24[nv1[k]] *	ncbot21[4] +
f24[nv2[k]] *	ncbot22[4] + f24[nv3[k]] *	ncbot23[4] +
f24[nv5[k]] *	ncbot25[4] + f24[nv6[k]] *	ncbot26[4];
ff25[k] = f25[k] *	ncbot20[5] + f25[nv1[k]] *	ncbot21[5] +
f25[nv2[k]] *	ncbot22[5] + f25[nv3[k]] *	ncbot23[5] +
f25[nv5[k]] *	ncbot25[5] + f25[nv6[k]] *	ncbot26[5];
ff26[k] = f26[k] *	ncbot20[6] + f26[nv1[k]] *	ncbot21[6] +
f26[nv2[k]] *	ncbot22[6] + f26[nv3[k]] *	ncbot23[6] +
f26[nv5[k]] *	ncbot25[6] + f26[nv6[k]] *	ncbot26[6];
ff27[k] = f27[k] *	* ncbot20[7] + f27[nv1[k]] * ncbot:	ncbot21[7] +
f27[nv2[k]] *	* ncbot22[7] + f27[nv3[k]] * ncbot:	ncbot23[7] +
f27[nv5[k]] *	* ncbot25[7] + f27[nv6[k]] * ncbot.	ncbot26[7];
ff28[k] = f28[k] *	ncbot20[8] + f28[nv1[k]] *	ncbot21[8] +
f28[nv2[k]] *	ncbot22[8] + f28[nv3[k]] *	ncbot23[8] +
f28[nv5[k]] *	ncbot25[8] + f28[nv6[k]] *	ncbot26[8];
<pre>break; case 2: ff10[k] = f10[k];</pre>		
<pre>ffll(k) = fll(k) * fll(nv3[k]) * fll(nv7[k]) *</pre>	* nctop10[1] + f11[nv1[k]] * nctop * nctop13[1] + f11[nv4[k]] * nctop * nctop17[1] + f11[nv8[k]] * nctop	nctop11[1] + nctop14[1] + nctop18[1];
ff12(k) = f12(k) *	* nctop10[2] + f12[nv1[k]] * nctop	nctop11[2] +
f12[nv3(k)] *	* nctop13[2] + f12[nv4[k]] * nctop	nctop14[2] +

Jun 20 1999 18:01 wet9is.c	Page 22
<pre>f12[nv5[k]] * ncleft15[2] + f12[nv2[k]] * ncleft12[2] + f12[nv4[k]] * ncleft14[2] + f12[nv8[k]] * ncleft18[2];</pre>	
<pre>ffl3[k] = f13[k] * ncleft10[3] + f13[nv1[k]] * ncleft11[3] + f13[nv5[k]] * ncleft15[3] + f13[nv2[k]] * ncleft18[3] + f13[nv4[k]] * ncleft14[3] + f13[nv8[k]] * ncleft18[3];</pre>	
<pre>ff14[k] = f14[k] * ncleft10[4] + f14[nv1[k]] * ncleft11[4] + f14[nv5[k]] * ncleft15[4] + f14[nv2[k]] * ncleft12[4] + f14[nv4[k]] * ncleft14[4] + f14[nv8[k]] * ncleft18[4];</pre>	
<pre>ff15[k] = f15[k] * ncleft10[5] + f15[nv1[k]] * ncleft11[5] + f15[nv5[k]] * ncleft15[5] + f15[nv2[k]] * ncleft12[5] + f15[nv4[k]] * ncleft14[5] + f15[nv8[k]] * ncleft18[5];</pre>	
<pre>ffl6[k] = f16[k] * ncleft10[6] + f16[nv1[k]] * ncleft11[6] + f16[nv5[k]] * ncleft15[6] + f16[nv2[k]] * ncleft12[6] + f16[nv4[k]] * ncleft14[6] + f16[nv8[k]] * ncleft18[6];</pre>	
<pre>ffl7(k) = f17(k) * ncleft10(7) + f17(nv1(k)) * ncleft11(7) + f17(nv5(k)) * ncleft15(7) + f17(nv2(k)) * ncleft12(7) + f17(nv4(k)) * ncleft14(7) + f17(nv8(k)) * ncleft18(7);</pre>	
<pre>ff18[k] = f18[k] * ncleft10[8] + f18[nv1[k]] * ncleft11[8] + f18[nv5[k]] * ncleft15[8] + f18[nv2[k]] * ncleft12[8] + f18[nv4[k]] * ncleft14[8] + f18[nv8[k]] * ncleft18[8];</pre>	
ff20[k] = f20[k];	
<pre>ff21[k] = f21[k] * ncleft20[1] + f21[nv1[k]] * ncleft21[1] + f21[nv5[k]] * ncleft25[1] + f21[nv2[k]] * ncleft22[1] + f21[nv4[k]] * ncleft24[1] + f21[nv8[k]] * ncleft28[1];</pre>	
<pre>ff22[k] = f22[k] * ncleft20[2] + f22[nv1[k]] * ncleft21[2] + f22[nv5[k]] * ncleft25[2] + f22[nv2[k]] * ncleft22[2] + f22[nv4[k]] * ncleft24[2] + f22[nv8[k]] * ncleft28[2];</pre>	
<pre>ff23[k] = f23[k] * ncleft20[3] + f23[nv1[k]] * ncleft21[3] + f23[nv5[k]] * ncleft25[3] + f23[nv2[k]] * ncleft22[3] + f23[nv4[k]] * ncleft24[3] + f23[nv8[k]] * ncleft28[3];</pre>	
<pre>ff24[k] = f24[k] * ncleft20[4] + f24[nv1[k]] * ncleft21[4] + f24[nv5[k]] * ncleft25[4] + f24[nv2[k]] * ncleft22[4] + f24[nv4[k]] * ncleft24[4] + f24[nv8[k]] * ncleft28[4];</pre>	
<pre>ff25[k] = f25[k] * ncleft20[5] + f25[nv1[k]] * ncleft21[5] + f25[nv5[k]] * ncleft25[5] + f25[nv2[k]] * ncleft22[5] + f25[nv4[k]] * ncleft24[5] + f25[nv8[k]] * ncleft28[5];</pre>	
<pre>ff26[k] = f26[k] * ncleft20[6] + f26[nv1[k]] * ncleft21[6] + f26[nv5[k]] * ncleft25[6] + f26[nv2[k]] * ncleft22[6] + f26[nv4[k]] * ncleft24[6] + f26[nv8[k]] * ncleft28[6];</pre>	
<pre>ff27(k] = f27(k] * ncleft20(7) + f27(nv1(k)) * ncleft21(7) + f27(nv5(k)) * ncleft25(7) + f27(nv2(k)) * ncleft22(7) + f27(nv4(k)) * ncleft24(7) + f27(nv8(k)) * ncleft28(7);</pre>	
<pre>ff28[k] = f28[k] * ncleft20[8] + f28[nv1[k]] * ncleft21[8] + f28[nv5[k]] * ncleft25[8] + f28[nv2[k]] * ncleft22[8] + f28[nv4[k]] * ncleft24[8] + f28[nv8[k]] * ncleft28[8];</pre>	
<pre>break; case 4: f10[k] = f10[k];</pre>	
<pre>ffll(k] = fll(k] * ncrightl0(1) + fll[nv3(k]) * ncrightl3[1] + fll[nv6(k]) * ncrightl6(1) + fll[nv2(k]) * ncrightl2[1] + fll[nv4(k]) * ncrightl4(l) + fll[nv7(k)] * ncrightl7[1];</pre>	

ec.utt.ro	
=	
_:	
_	
-:	
Q	
Φ.	
_	
_	
wi.	
numaga	
ပ္	
w.	
_	
⊑	
_	
=	
_	
Ε.	
=	
μ	
=	
_	
œ.	
Φ	
_	
ᅐ	
sotone	
0	
ຶດ	
<u></u>	
Ω	
_	
Ç	
Φ.	
=	
Frinted	
⋤	
1	
_	

wet9is.c

Jun 20 1999 18:01		Š	Wetsis.c			cz aße zo	Juli 20 1999 10.01
ff12[k] = f12[k] = f12[[k] = f12[k] * ncrig f12[nv6[k]] * ncrig f12[nv4[k]] * ncrig	ncright10[2] ncright16[2] ncright14[2]	+ f12[nv3[k]] + f12[nv2[k]] + f12[nv7[k]]	K * * *	ncright13[2] + ncright12[2] + ncright17[2];		
ff13[k] = f13[f13[= f13[k] * ncrig 3[nv6[k]] * ncrig 3[nv4[k]] * ncrig	ncright10[3] ncright16[3] ncright14[3]	+ f13[nv3[k]] + f13[nv2[k]] + f13[nv7[k]]	* * *	ncright13[3] + ncright12[3] + ncright17[3];		on la
ff14[k] = f14[= f14[k] * ncrig 4[nv6[k]] * ncrig 4[nv4[k]] * ncrig	ncright10[4] ncright16[4] ncright14[4]	+ f14[nv3[+ f14[nv2[+ f14[nv7]	* * * EEE	ncright13[4] + ncright12[4] + ncright17[4];		
ff15[k] = f15[r	£15[k] * 1v6[k]] * 1v4[k]] *	ncright10[5] ncright16[5] ncright14[5]	+ f15[nv3[k]] + f15[nv2[k]] + f15[nv7[k]]	* * *	ncright13[5] + ncright12[5] + ncright17[5];		9 9
ff16[k] = f16[= f16[k] * ncrig 6[nv6[k]] * ncrig 6[nv4[k]] * ncrig	ncright10[6] ncright16[6] ncright14[6]	+ f16[nv3[+ f16[nv2[+ f16[nv7[XXX XXX	ncright13[6] + ncright12[6] + ncright17[6];		
ff17[k] = f17[k] f17[nv6[k]] f17[nv4[k]]	* * *	ncright10[7] ncright16[7] ncright14[7]	+ £17[nv3] + £17[nv2] + £17[nv7]	* * *	ncright13[7] + ncright12[7] + ncright17[7];		110[K] = 1110[K] (ecprod]
ff18[k] = f18 f18 f18	= f18[k] * ncrig [nv6[k]] * ncrig [nv4[k]] * ncrig	ncright10[8] ncright16[8] ncright14[8]	+ f18[nv3[+ f18[nv2[+ f18[nv7[* * *	ncright13[8] + ncright12[8] + ncright17[8];	ALAPUT.	
ff20[k] *	= f20[k];						(ecprod f21[k] = ff21[)
ff21[k] = f21 f21 f21	[k] = f21[k] * ncriq f21[nv6[k]] * ncriq f21[nv4[k]] * ncriq	ncright20[1] ncright26[1] ncright24[1]	+ f21[nv3[k]] + f21[nv2[k]] + f21[nv7[k]]	* * *	ncright23[1] + ncright22[1] + ncright27[1];		f22[k] = ff22[cd] (ecprod) [623[k] = ff23[k]
ff22[k] = f22 f22 f22	k] = f22[k] * ncrig f22[nv6[k]] * ncrig f22[nv4[k]] * ncrig	ncright20[2] ncright26[2] ncright24[2]	+ f22[nv3[k]] + f22[nv2[k]] + f22[nv7[k]]	* * *	ncright23[2] + ncright22[2] + ncright27[2];		f24[k] = ff24[k] (ecprod) (ecprod) [f25[k] = ff25[k]
ff23[k] = f23 f23	k] = £23[k] * ncrig £23[nv6[k]] * ncrig £23[nv4[k]] * ncrig	ncright20[3] ncright26[3] ncright24[3]	+ f23[nv3[k]] + f23[nv2[k]] + f23[nv7[k]]	* * *	ncright23[3] + ncright22[3] + ncright27[3];	*****	f26[k] = ff26[k] (ecprod) (27[k] = ff27[k]
ff24[k] * f24 f24	k] = £24[k] * ncriç £24[nv6[k]] * ncriç £24[nv4[k]] * ncriç	ncright20[4] ncright26[4] ncright24[4]	+ f24[nv3[k]] + f24[nv2[k]] + f24[nv7[k]]	* * *	ncright23[4] + ncright22[4] + ncright27[4];		f28[k] = ff28[k] (ecprod
ff25[k] = f25 f25 f25	[k] = £25[k] * ncriq £25[nv6[k]] * ncriq £25[nv4[k]] * ncriq	ncright20[5] ncright26[5] ncright24[5]	+ f25[nv3[k]] + f25[nv2[k]] + f25[nv7[k]]	* * * (K)	ncright23[5] + ncright22[5] + ncright27[5];		f10[k] = ff10[k] (ecprod)
ff26[k] = f26 f26 f26	f26[k] * nv6[k]] * nv4[k]] *	ncright20[6] ncright26[6] ncright24[6]	+ f26[nv3[k]] + f26[nv2[k]] + f26[nv7[k]]	* * *	ncright23[6] + ncright22[6] + ncright27[6];		f12[k] = ff12[k] = ff12[l] (ecprod)
ff27(k) = f27 f27	<pre>(k) = £27(k) * ncriq f27[nv6[k]] * ncriq f27[nv4[k]] * ncriq</pre>	ncright20[7] ncright26[7] ncright24[7]	+ f27[nv3[k]] + f27[nv2[k]] + f27[nv7[k]]	* * *	ncright23[7] + ncright22[7] + ncright27[7];		f14[k] = f114[k] = f115[k] = f115[k]
ff28[k] * f28 f28 f28	= f28[k] * ncriq [nv6[k]] * ncriq [nv4[k]] * ncriq	ncright20[8] ncright26[8] ncright24[8]	+ f28[nv3[+ f28[nv2[+ f28[nv7[K []	* ncright23[8] + * ncright22[8] + * ncright27[8];		f16[k] = ff16[k] = ff16[ff16] = ff17[f] = ff17[ff17] = ff
break;	^					•	f18[k] = ff18[] (ecprod
-						,	f20[k] = ff20[)
void ifdprop(void)							£21[k] = ££21[]

		k] + csforce_y * uyloc[k];	<pre>aul * (ff10[k] - neq10[k]) + prodscal) * neq10[k]; prodscal * neq11[k] + prodscal * neq11[k] + aul * (ff12[k] - neq12[k]) +</pre>	<pre>prodscal) * neq12 k]; au1 * (ff13 k] - neq13 k]) + prodscal) * neq13 k]; au1 * (ff14 k] - neq14 k]) +</pre>		.au1 * (ff16[k] - neq16[k]) + prodscal) * neq16[k] ,)	auz * (ff20[k] - neg20[k]) + brodscal) * neg2[k]:	au2 * ($ff21[k]$ - $neq21[k]$) + prodscal) * $neq21[k]$; au2 * ($fe22[k]$ - $neq2[k]$) +	* neq22[k]; f23[k] - neq23[k])	ff24[k]; ff24[k] - neq24[k]) +)) * nec24[k]	f25[k] - * neq25[$\{ \{ \{ \{ \{ \{ \} \} \} \} \} \} \} $ $\{ \{ \{ \{ \} \} \} \} \} \} \} $ $\{ \{ \{ \{ \} \} \} \} \} \} \} \} \} \} \} \} \} \} \} \}$	* neg27 f28[k] - * neg28	(f10[k] _ neg10[k]) +	(11[k] - neq10[k];	fil(k) - neq12(k) +		[k] - neq14[k] neq14[k]	au1 * (ii)k - neqi5 k) + prodscal) * neqi5 k ; au1 * (ii6 k - neqi6 k) +			au2 * (f20[k] - neq20[k]) + prodscal * neq20[k];
int k; double prodscal;	<pre>for (k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>	prodscal = csforce_x * uxloc[k]	8	1010	1 0 1	0 1 0	<pre>(ecprod1[7] - prodscal) f18[k] = ff18[k] - ctaul * (f (ecprod1[8] - prodscal)</pre>	f20[k] = ff20[k] - ctau2 * (g 1 g	(ecprod2[2] - = ff23[k] - ct	<pre>(ecprod2[3] - prodscal) f24[k] = ff24[k] - ctau2 * (f (ecprod2[4] - prodscal)</pre>	= ff25[k] - ctau2 * (ecprod2[5] - prodsca	ਰ _। ਹ	(ecprod2[7] - = ff28[k] - ct (ecprod2[8] -	ctaul *	ctaul *	f12[k] = ff12[k] - ctaul * (f12[k]	ctaul *	= ff14[k] - ctaul * (ecprod1[4] - prodsc	ctaul * - prodsc ctaul *	(ecprod1[6] - prodsc = ff17[k] - ctaul *	<pre>(ecprod1[7] - prodscal) f18[k] = ff18[k] - ctaul * (fi (ecprod1[8] - prodscal)</pre>	f20[k] = ff20[k] - ctau2 * (f20] coprod2[0] - prodscal) f20 f20

Printed by sofonea from flumag3.mec.utt.ro

f22[k] = ff22[k] - ctau2 *	ctau2 * (£22[k] - neq22[k]) + - prodscal) * neq22[k]) + - prodscal) * neq22[k]) + - trodscal) * neq23[k]) + - prodscal) * neq24[k]) + - prodscal) * neq24[k]) + - prodscal) * neq24[k]) + - prodscal) * neq26[k]) + - prodscal) * neq25[k]) + - prodscal) * neq26[k]) + - prodscal) * neq28[k]) + - prodscal) * neq28[k]) + - prodscal) * neq28[k]) +	
=	2 [K]; neq22[K]) neq23[K]; neq23[K]; neq25[K]; neq25[K]; neq25[K]; neq26[K]; neq26[K]; neq27[K]; neq27[K];	
	neq23[K]) 33[K]; neq24[K]) 44[K]; neq25[K]) 55[K]; neq26[K]) neq27[K])	
	74 [k] 74 [k] 74 [k] 75	
(ecprodz[1])	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
(ecprod2[5])	prodecal) * neq25 K]; au2 * (f26[k] - neq26[k]) prodecal) * neq26[k]; au2 * (f27[k] - neq27[k]) prodecal) * neq27[k]; au3 * (f28[k] - neq28[k])	
(ecprod2[6] 7[K] = ff27[K] - (ecprod2[7]) - (ecprod2[7]) - (ecprod2[8]) - (ecprod2[8]) - (k=0) (k=0) (k=0) (printf("iter=%d printf("iter=%d p		
<pre>61kj = riz8lkj - cf (ecprod2[8] - (k==0) printf("iter=%d k printf("iter=%d k</pre>	* 128 K = Hedzolk	
// /f(k==0) printf("iter=%d k: printf("iter=%d k: // // // // // // // // // // // // //	scal) * neq28[k];	•
<pre>if(k==0) printf("iter=%d k: printf("iter=%d k: f10[k]-neq1 */</pre>		
printf("iter=%d k° printf("iter=%d k° fl0[k]-negil */		
f10[k]-neq1	==qq iii)==ee iizv==e\n'.iter,k,iiiv\r,iizv\r,); ==qd neq10==ee f10-neq==e\n'.iter,k,neq10[k],	
	0[k]);	
void iupfd(void)		
K <nnodes_att;< td=""><td>K++)</td><td></td></nnodes_att;<>	K++)	
ff10[k] = f10[k];		
	null[1] + fll[nv3[nv3[k]]] * nul3[l] + fll[nv3[nv3[k]]] * nul3[l] + fll[nv6[k]] * nul2[l] +	
* * *		
* *	+ + f12[nv4[nv4[k]]] * nu14[2]	
* *	+ fig[nv8[k]]	
	222	
* *	+ + f13[nv1[nv1[k]]] * nu11[3]	
£13[nv4[k]] *		
	9.00	
* *	+ + f14[nv2[nv2[k]]] * nu12[4]	
* *	+ £14[nv5[k]] +	
f14[nv3[k]] * n f14[nv5[k]] * n	nul7[4] + nul3[4] + f14[nv2[nv5[k]]] * nul6[4];	
* *	5] + 5] + f15[nv3[nv3[k]]] * nu16[5]	,
f15[nv4[k]] * n f15[nv3[nv7[k]]] * n f15[nv4[nv7[k]]] * n	null[5] + f15[nv7[k]] * nulv][5] + nulv[5] + nulv[5] + f15[nv4[nv4[k]]] * nulv[6] + nulv[6] + f15[nv7[kv]]] * nulv[6];	•
ff16[k] = f16[k] * n f16[nv1[k]] * n	nu16[6] + f16[nv1[nv1[k]]] * nu15[6] +	

Jun 20 1999 18:01	wet9is.c	à	age 26
<pre>f16[nv1[nv8[k]]] * nul1[6] + f16[nv4[nv8[k]]] * nul4[6] +</pre>	f16[nv4[nv4[k]]] f16[nv8[nv8[k]]]	* nu17[6] + * nu18[6];	
ffl(k) = fl(k) * nul(7) + fl(nul(k)) * nul(7) + fl(nul(k)) * nul(3) + fl(nul(nul(k)) * nul(1) + fl(nul(nul(k))) * nul(7) +	f17[nv1[nv1[k]]] f17[nv5[k]] f17[nv2[nv2[k]]] f17[nv5[nv5[k]]]	* nul8[7] + * nul0[7] + * nul6[7] + * nul5[7];	
[k] = f18[k] f18[nv3[k]] f18[nv2[k]] nv3[nv6[k]]] nv2[nv6[k]]]	f18[nv3[nv3[k]]] f18[nv6[k]] f18[nv2[k]]] f18[nv6[nv6[k]]]	* nu17[8] + * nu10[8] + * nu15[8] + * nu15[8];	
ff20[k] = f20[k]; ff21[k] = f21[k] * nu21[1] + f21[nv3[k]] * nu26[1] + f21[nv2[k]] * nu26[1] + f21[nv2[k]] * nu26[1] + f21[nv4[k]] * nu26[1] + f21[nv4[k]] * nu26[1] +	f21[nv3[nv3[k]]] f21[nv6[k]] f21[nv7[k]]	* nu23[1] + * nu22[1] + * nu24[1] +	***
ff22[k] = f22[k] * nu22[2] f22[nv4[k]] * nu20[2] f22[nv1[k]] * nu25[2] f22[nv1[k]] * nu26[2] f22[nv3[k]] * nu26[2] f22[nv7[k]] * nu26[2]	+ f22[nv4[nv4[k]]] + f22[nv8[k]] + + + f22[nv4[k]]]	* nu24[2] + * nu21[2] + * nu27[2];	
ff23[k] = f23[k] * nu23[3] + f23[nv1[k]] * nu20[3] + f23[nv1[k]] * nu27[3] + f23[nv2[k]] * nu28[3] - f23[nv2[k]] * nu26[3] +	+ f23[nv1[nv1[k]]] + f23[nv8[k]] + f23[nv1[nv5[k]]] + f23[nv1[nv5[k]]]	* nu21[3] + * nu24[3] + * nu25[3];	
ff24[k] = f24[k] * nu24[4] - f24[nv2[k]] * nu20[4] - f24[nv1[k]] * nu28[4] - f24[nv3[k]] * nu25[4] - f24[nv3[k]] * nu27[4] - f24[nv5[k]] * nu23[4] -	+ f24[nv2[nv2[k]]] + f24[nv5[k]] + + + + + f24[nv2[nv5[k]]]	* nu22[4] + * nu21[4] + * nu26[4];	
ff25[R] = f25[K] * nu25[5] f25[nv3[K]] * nu22[5] f25[nv4[K]] * nu21[5] f25[nv4[k]] * nu23[5] f25[nv4[k]]] * nu23[5]	+ f25[nv3[nv3[k]]] + f25[nv7[k]] + f25[nv4[nv4[k]]] + f25[nv7[nv7[k]]]	* nu26[5] + * nu20[5] + * nu28[5] + * nu28[5] +	
<pre>ff26[k] = f26[k] * nu26[6] f26[nv1[k]] * nu22[6] f26[nv4[k]] * nu21[6] f26[nv4[k]]] * nu21[6] f26[nv4[nv8[k]]] * nu21[6]</pre>	+ f26[nv1[nv1[k]]] + f26[nv8[k]] + f26[nv4[nv4[k]]] + f26[nv8[nv8[k]]]	* nu25[6] + * nu20[6] + * nu27[6] + * nu28[6];	
ff27[k] = f27[k] * nu27[7] f27[nv1[k]] * nu24[7] f27[nv2[k]] * nu23[7] f27[nv2[k]] * nu21[7] f27[nv2[nv5[k]]] * nu22[7]	+ + £27[nv1[hv1[k]]] + £27[nv5[k]] + £27[nv2[nv5[k]]] + £27[nv5[nv5[k]]]	* nu28(7] + * nu20(7] + * nu26(7] + * nu25(7);	
ff28[k] = f28[k] * nu28[8] f28[nv3[k]] * nu24[8] f28[nv2[k]] * nu21[8] f28[nv2[k]] * nu21[8] f28[nv2[nv6[k]]] * nu23[8]	+ f28[nv3[nv3[k]]] + f28[nv6[k]] + f28[nv2[k]]] + f28[nv6[nv6[k]]]	* nu27[8] + * nu20[8] + * nu25[8] + * nu26[8];	

m flumag3.mec.utt.ro
Ē
sofonea
δ
Printed

Jun 20 1999 18:01	1 age 21
/* if(k==0) printf("iter=%d k=%d fl=%e f2=%e\n",iter,k,fl0[k],f20[k]);	
void iser(void)	
<pre>int k; for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>	
<pre>switch(boundary_mode(k])</pre>	
[case 0: ffl0[k] = fl0[k];	
<pre>ff11[k] = f11[k] * nsltr[1] + f11[nv3[k]] * nslt1[1] + f11[nv7[k]] * nsltn[1] + f11[nv4[k]] * nsltr[1];</pre>	
ff12[k] = f12[k] * ns1t1[2] + f12[nv4[k]] * ns1b1[2] + f12[nv8[k]] * ns1br[2];	
<pre>ff13[k] = f13[k] * ns1b1[3] + f13[nv1[k]] * ns1br[3] + f13[nv2[k]] * ns1t1[3] + f13[nv5[k]] * ns1tr[3];</pre>	
<pre>ff14[k] = f14[k] * nslbr[4] + f14[nv3[k]] * nslbl[4] +</pre>	
ff15[k] = f15[k] * nsltr[5] + f15[nv3[k]] * nsltl[5] + f15[nv4[k]] * nsltr[5];	
<pre>ff16[k] = f16[k] * nsltl[6] + f16[nv4[k]] * nslbl[6] + f16[nv1[k]] * nsltr[6] + f16[nv8[k]] * nslbr[6];</pre>	
ff17[k] = f17[k] * nslb1[7] + f17[nv1[k]] * nslbr[7] + f17[nv2[k]] * nslt1[7] + f17[nv5[k]] * nsltr[7];	
<pre>file[k] = f18[k] * nslbr[8] + f18[nv2[k]] * nslb1[8] + f18[nv2[k]] * nsltr[8] + f18[nv6[k]] * nslt1[8];</pre>	
ff20[k] = f20[k];	-
ff21[k] = f21[k] * ns2tx[l] + f21[nv3[k]] * ns2t1[l] + f21[nv4[k]] * ns2bx[l];	
ff22[k] = f22[k] * ns2t1[2] + f22[nv4[k]] * ns2b1[2] + f22[nv8[k]] * ns2br[2];	
ff23[k] = f23[k] * ns2b1[3] + f23[nv1[k]] * ns2br[3] + f23[nv2[k]] * ns2tr[3];	- environment
<pre>ff24[k] = f24[k] * ns2br[4] + f24[nv3[k]] * ns2bl[4] +</pre>	
ff25[k] = f25[k] * ns2tr[5] + f25[nv3[k]] * ns2tl[5] + f25[nv4[k]] * ns2br[5];	
<pre>ff26[k] = f26[k] * ns2t1[6] + f26[nv4[k]] * ns2b1[6] +</pre>	
<pre>ff27[k] = f27[k] * ns2b1[7] + f27[nv1[k]] * ns2br[7] +</pre>	
<pre>ff28[k] = f28[k] * ns2br[8] + f28[nv3[k]] * ns2b1[8] + break;</pre>	

Jun 20 1999 18:01	wet9is.c	Page 28
££10[k] = £10[k];		
ff11[k] = f11[k] f11[nv6[k]]] * nsblbr[1] + f11[nv3[k]] * nsblb1[1] +] * nsblt1[1] + f11[nv2[k]] * nsbltr[1];	
ff12[k] = f12[k] f12[nv2[k]]] * nsblb][2] + f12[nv1[k]] * nsblbr[2] +] * nsblt1[2] + f12[nv5[k]] * nsbltr[2];	
ff13[k] = f13[k] f13[nv2[k]]	* nslb1(3] + f13(nv1(k)] * nslbz[3] + * nslt1[3] + f13(nv5[k)] * nsltz[3];	
<pre>ff14[k] = f14[k] f14[nv2[k]]</pre>	* nslbr[4] + f14[nv3[k]] * nslbl[4] + * nsltr[4] + f14[nv6[k]] * nsltl[4];	
ff15[k] = f15[k] f15[nv2[k]]] * nsblb1[5] + f15[nv1[k]] * nsblbr[5] +] * nsblt1[5] + f15[nv5[k]] * nsbltr[5];	
ff16[k] = f16[k] f16[nv2[k]]] * nsblbr[6] + £16[nv3[k]] * nsblbl[6] +] * nsbltr[6] + £16[nv6[k]] * nsbltl[6];	
ff17[k] = f17[k] f17[nv2[k]]	* nsibl[7] + f17[nv1[k]] * nsibr[7] + * nsit1[7] + f17[nv5[k]] * nsitz[7];	
ff18[k] = f18[k] f18[nv2[k]]	* nslbr[8] + f18[nv3[k]) * nslb1[8] + * nsltr[8] + f18[nv6[k]] * nslt1[8];	
ff20[k] = f20[k];		
ff21(k) = f21[k] f21[nv6[k]]	* nsb2br[1] + £21[nv3[k]] * nsb2b1[1] + * nsb2t1[1] + £21[nv2[k]] * nsb2tr[1];	
ff22[k] = f22[k]	* nsb2b1[2] + £22[nv1[k]] * nsb2br[2] + * nsb2t1[2] + £22[nv5[k]] * nsb2tr[2];	
ff23[k] = f23[k] f23[nv2[k]]] * ns2bl[3] + f23[nv1[k]] * ns2br[3] +] * ns2tl[3] + f23[nv5[k]] * ns2tr[3];	
ff24[k] = f24[k] f24[nv2[k]]] * ns2br[4] + f24[nv3[k]] * ns2b1[4] +] * ns2tr[4] + f24[nv6[k]] * ns2t1[4];	
ff25[k] = f25[k] f25[nv2[k]]] * nsb2b1[5] + £25[nv1[k]] * nsb2br[5] +] * nsb2t1[5] + £25[nv5[k]] * nsb2tr[5];	
ff26[k] = f26[k] f26[nv2[k]]] * nsb2br[6] + f26[nv3[k]] * nsb2b1[6] +] * nsb2tr[6] + f26[nv6[k]] * nsb2t1[6];	,
ff27[k] = f27[k] f27[nv2[k]]] * ns2b1[7] + £27[nv1[k]] * ns2bz[7] +] * ns2t1[7] + £27[nv5[k]] * ns2tz[7];	
[f28[k] = f28[k] f28[my2[k]]] * ns2br[8] + f28[nv3[k]] * ns2bl[8] + + + + + + + + + + + + + + + + + +	
break;	[[v]oar]ost [o]tass	
<pre>case 2: ff10[k] = f10[k];</pre>		
ff11[k] = f11[k] f11[nv7[k]]] * nsltr[1] + fll[nv3[k]] * nsltl[1] +] * nslbl[1] + fll[nv4[k]] * nslbr[1];	
ff12[k] = f12[k] f12[nv1[k]]] * nsltl[2] + f12[nv4[k]] * nslb1[2] +] * nsltr[2] + f12[nv8[k]] * nslbr[2];	
ff13[k] = f13[k] f13[nv4[k]]] * nstltl[3] + f13[nv1[k]] * nstltx[3] +] * nstlbl[3] + f13[nv8[k]] * nstlbx[3];	
<pre>ff14[k] = f14[k] f14[nv4[k]]</pre>] * nstltr[4] + £14[nv3[k]] * nstltl[4] +] * nstlbr[4] + £14[nv7[k]] * nstlbr[4];	
ff15[k] = f15[k]] * nsltr[5] + f15[nv3[k]] * nsltl[5] +	

Wet9is.c

28

4	-	
	رين	
- 1	n	

ff21[k] = f21[k] * nel2t1[1] + f21[nv1[k]] * nel2tr[1] +

ff20[k] = f20[k];

Jun 20 1999 18:01 wet9ls.c Page	je 29	Jun 20 1999 18:01 wet91s.
f15[nv7[k]] * nslb1[5] + f15[nv4[k]] * nslbr[5];		£21[nv4[k]] * nsl2b1[1] + £2
<pre>ff16[k] = f16[k] * ns1t1[6] + f16[nv4[k]] * ns1b1[6] + f16[nv1[k]] * ns1tr[6] + f16[nv8[k]] * ns1br[6];</pre>		ff22[k] = f22[k] * ns21[2] + f22 f22[nv1[k]] * ns2tr[2] + f22
ff17[k] = f17[k] * nstltr[7] + f17[nv3[k]] * nstlt1[7] + f17[nv7[k]] * nstlb1[7];		ff23[k] = f23[k] * ns2b1[3] + f23 f23[nv2[k]] * ns2t1[3] + f23
<pre>ff18[k] = f18[k] * nst1t1[8] + f18[nv1[k]] * nst1tr[8] + f18[nv4[k]] * nst1b1[8] + f18[nv8[k]] * nst1br[8];</pre>		ff24[k] = f24[k] * nsl2bl[4] + f2 f24[nv2[k]] * nsl2tl[4] + f2
		ff25[k] = f25[k] * ns12b1[5] + f2 f25[nv2[k]] * ns12t1[5] + f2
<pre>ff21[k] = f21[k] * ns2tr[1] + f21[nv3[k]] * ns2tr[1] + f21[nv7[k]] * ns2b1[1] + f21[nv4[k]] * ns2br[1];</pre>		* ns2t1[6] +
<pre>ff22[k] = f22[k] * ns2tl[2] + f22[nv4[k]] * ns2bl[2] + f22[nv1[k]] * ns2tr[2] + f22[nv8[k]] * ns2br[2];</pre>		* ns2b1[7] +
<pre>ff23[k] = f23[k] * nst2t1[3] + f23[nv1[k]] * nst2tr[3] + f23[nv4[k]] * nst2b1[3] + f23[nv8[k]] * nst2br[3];</pre>		* nsl2t1[8] -
<pre>ff24[k] = f24[k] * nst2tr[4] + f24[nv3[k]] * nst2t1[4] + f24[nv4[k]] * nst2br[4] + f24[nv7[k]] * nst2bl[4];</pre>		[9]TOZTSU *
<pre>ff25[k] = f25[k] * ns2tr[5] + f25[nv3[k]] * ns2tr[5] + f25[nv7[k]] * ns2b1[5] + f25[nv4[k]] * ns2br[5];</pre>		<pre>case 4: ff10[k] = f10[k];</pre>
<pre>ff26[k] = f26[k] * ns2t1[6] + f26[nv4[k]] * ns2t1[6] + f26[nv1[k]] * ns2tr[6] + f26[nv8[k]] * ns2tr[6];</pre>		ffll[k] = fll[k] * nsltr[l] + fll fll[nv7[k]] * nslbl[l] + fll
<pre>ff27[k] = f27[k] * nst2tr[7] + f27[nv3[k]] * nst2t1[7] + f27[nv4[k]] * nst2br[7] + f27[nv7[k]] * nst2b1[7];</pre>		ffl2[k] = f12[k] * nsrltr[2] + f7 f12[nv4[k]] * nsrlbr[2] + f1
<pre>ff28[k] = f28[k] * nst2tl[8] + f28[nv1[k]] * nst2tr[8] + f28[nv4[k]] * nst2b1[8] + f28[nv8[k]] * nst2br[8];</pre>		ff13[k] = f13[k] * nsrlbr[3] + f1 f13[nv2[k]] * nsrltr[3] + f1
break;		ff14[k] = f14[k] * nslbr[4] + f14 f14[nv2[k]) * nsltr[4] + f14
case 3: ff10[k] = f10[k];		* nsitr[5] + * nsitr[5] +
<pre>ff11[k] = f11[k] * nsllt1[1] + f11[nv1[k]] * nslltr[1] + f11[nv4[k]] * nsllb1[1] + f11[nv8[k]] * nsllbr[1];</pre>		* nsrlbr[6]
<pre>fil2[k] = f12[k] * nsltl[2] + f12[nv4[k]) * nslb1[2] + f12[nv1[k]] * nsltr[2] + f12[nv8[k]] * nslbr[2];</pre>		* nsrltr[7]
<pre>ff13[k] = f13[k] * nslb1[3] + f13[nv1[k]] * nslbr[3] + f13[nv2[k]] * nslt1[3] + f13[nv5[k]] * nsltr[3];</pre>		* nslbr[8] + f
<pre>ff14[k] = f14[k] * hsllbl[4] + f14[nv1[k]] * nsllbr[4] + f14[nv2[k]] * nslltl[4] + f14[nv5[k]] * nslltr[4];</pre>		nsitr[8] +
<pre>ff15[k] = f15[k] * nsllbl[5] + f15[nv1[k]] * nsllbr[5] + f15[nv2[k]] * nslltl[5] + f15[nv5[k]] * nslltr[5];</pre>		ff21[k] = f21[k] * ns2tr[1] + f2: f21[nv7[k]] * ns2b1[1] + f2:
<pre>ff16[k] = f16[k] * nslt1[6] + f16[nv4[k]] * nslb1[6] + f16[nv8[k]] * nsltr[6] + f16[nv8[k]] * nslbr[6];</pre>		ff22[k] = f22[k] * nsr2tr[2] + f; f22[nv4[k]] * nsr2br[2] + f5
<pre>ff17[k] = f17[k] * ns1b1[7] + f17[nv1[k]] * ns1bz[7] + f17[nv2[k]] * ns1t1[7] + f17[nv5[k]] * ns1tz[7];</pre>		ff23[k] = f23[k] * nsr2br[3] + f; f23[nv2[k]] * nsr2tr[3] + f
	-	

lumag3.mec.utt.ro	
from f	
y sofonea i	
Printed b	

Jun 20 1999 18:01	*	wet9is.c	Page 30
	f21[nv4[k]] * nsl2b1[1]	[1] + f21[nv8[k]] * ns12br[1];	
ff22[k]	= f22[k] * ns2t1[2] f22[nv1[k]] * ns2tr[2]	<pre>!] + f22[nv4[k]] * ns2b1[2] + !] + f22[nv8[k]] * ns2br[2];</pre>	
ff23[k]	= f23[k] * ns2b1[3] f23[nv2[k]] * ns2t1[3]	<pre>3] + f23[nv1[k]] * ns2br[3] + 3] + f23[nv5[k]] * ns2tr[3];</pre>	
ff24[k]	= f24[k] * nsl2b1[4] f24[nv2[k]] * nsl2t1[4]	<pre>[4] + f24[nv1[k]] * nsl2br[4] + [4] + f24[nv5[k]] * nsl2tr[4];</pre>	
ff25[k]	= f25[k] * nsl2bl[5] f25[nv2[k]] * nsl2tl[5]	[5] + f25[nv1[k]] * nsl2br[5] + [5] + f25[nv5[k]] * nsl2tr[5];	
ff26[k]	= f26[k] * ns2t1[6] f26[nv1[k]] * ns2tr[6]	5] + f26[nv4[k]] * ns2b1[6] + 5] + f26[nv8[k]] * ns2br[6];	
ff27[k]	= f27[k] * ns2b1[7] f27[nv2[k]] * ns2t1[7]	<pre>// + f27[nv1[k]] * ns2br[7] + // + f27[nv5[k]] * ns2tr[7];</pre>	
ff28[k]	= f28[k] * ns12t1[8] f28[nv4[k]] * ns12b1[8]	[8] + f28[nv1[k]] * ns12tr[8] + [8] + f28[nv8[k]] * ns12br[8];	
break;			
case 4: ff10[k]	= f10[k];		
ff11[k]	= fll[k] * nsltr[l] fll[nv7[k]] * nslbl[l]	[] + fll[nv3[k]] * nsltl[l] + [] + fll[nv4[k]] * nslbr[l];	
££12[k]	= f12[k] * nsrltr[2] f12[nv4[k]] * nsrlbr[2]	[2] + f12[nv3[k]] * nsrlt1[2] + [2] + f12[nv7[k]] * nsrlb1[2];	
ff13[k]	= f13[k] * nsrlbr[3] f13[nv2[k]] * nsrltr[3]	[3] + f13[nv3[k]] * nsrlb1[3] + [3] + f13[nv6[k]] * nsrlt1[3];	
ff14[k]	= f14[k] * nslbr[4] f14[nv2[k]] * nsltr[4]	4] + f14[nv3[k]] * ns1b1[4] + 4] + f14[nv6[k]] * ns1t1[4];	
ff15[k]	= f15[k] * nsltr[5] f15[nv7[k]] * nslb1[5]	5] + f15[nv3[k]] * ns1t1[5] + 5] + f15[nv4[k]] * ns1bx[5];	
ff16[k]	= f16[k] * nsrlbr[6] f16[nv2[k]] * nsrltr[6]	<pre>[6] + f16[nv3[k]] * nsr1b1[6] + [6] + f16[nv6[k]] * nsr1t1[6];</pre>	
ff17[k]	= f17[k] * nsrltr[7] f17[nv4[k]] * nsrlbr[7]	[7] + f17[nv3[k]] * nsr1t1[7] + [7] + f17[nv7[k]] * nsr1b1[7];	
ff18[k]	= f18[k] * nslbr[8] f18[nv2[k]] * nsltr[8]	8] + f18[nv3[k]] * ns1b1[8] + 8] + f18[nv6[k]] * ns1t1[8];	
ff20[k]	= f20[k];		
ff21[k]	= f21[k] * ns2tr[1] f21[nv7[k]] * ns2b1[1]	<pre>1] + f21[nv3[k]] * ns2t1[1] + 1] + f21[nv4[k]] * ns2br[1];</pre>	
ff22[k]	= f22[k] * nsr2tr[2] f22[nv4[k]] * nsr2br[2]	<pre>[2] + f22[nv3[k]] * nsr2t1[2] + [2] + f22[nv7[k]] * nsr2b1[2];</pre>	
ff23[k]	= f23[k] * nsr2br[3] f23[nv2[k]] * nsr2tr[3]	<pre>[3] + f23[nv3[k]] * nsr2b1[3] + [3] + f23[nv6[k]] * nsr2t1[3];</pre>	
ff24[k]	= f24[k] * ns2br[4] f24[nv2[k]] * ns2tr[4]	4] + f24[nv3[k]] * ns2b1[4] + 4] + f24[nv6[k]] * ns2t1[4];	
££25[k]	= f25[k] * ns2tr[5] f25[nv7[k]] * ns2b1[5]	5] + f25[nv3[k]] * ns2t1[5] + 5] + f25[nv4[k]] * ns2br[5];	

6	
ij	
ပ္က	
Ĕ	
က္က	
ğ	
₫	
Ξ	
2	
fonea from flumag3.mec.utt.ro	
Ĕ	
ğ	
S ≥	
0	
ğ	
Printed by sof	

Jun 20 1999 18:01	Wetgistc	Lage 31	
[£26[k] =	f26[k] * nsr2br[6] + f26[nv3[k]] * nsr2bl[6] + f26[nv2[k]] * nsr2tr[6] + f26[nv6[k]] * nsr2tl[6];		
ff27[k] =	f27[k] * nsr2tr[7] + f27[nv3[k]] * nsr2t1[7] + f27[nv4[k]] * nsr2b1[7];		
ff28[k] =	f28[k] * ns2br[8] + f28[nv3[k]] * ns2b1[8] + f28[nv2[k]] * ns2t1[8];		
break;			
-			
void ilin(void)			
int k; for(k=0; k <nnodes_all; k++)<="" td=""><td>11; k++)</td><td></td><td></td></nnodes_all;>	11; k++)		
{ switch(boundary_mode[k])	_mode [k])		
case 0: ff10[k] ==	f10[k];		
ff11(k) =	<pre>f11(k) * nltr[1] + f11[nv3(k]] * nltr[1] + f11[nv7(k]] * nltb1[1] + f11[nv4(k]] * nltbr[1];</pre>		
ff12[k] =	f12[nv1[k]] * nlit1[2] + f12[nv4[k]] * nlib1[2] + f12[nv4[k]] * nlibr[2];	-	
ff13[k] =	f13[k] * nlib1[3] + f13[nv1[k]] * nlibx[3] + f13[nv2[k]] * nlitx[3];		
ff14[k] =	f14[k] * nllbr[4] + f14[nv3[k]] * nllbl[4] + f14[nv2[k]] * nlltr[4] + f14[nv6[k]] * nlttl[4];		
ff15[k] =	f15[k] * nlltr[5] + f15[nv3[k]] * nlltl[5] + f15[nv7[k]] * nllbl[5] + f15[nv4[k]] * nllbr[5];		
ff16[k] =	f16[k] * nlltl[6] + f16[nv4[k]] * nllbl[6] + f16[nv4[k]] * nllbr[6];		
££17[k] =	f17[k] * nllbl[7] + f17[nv1[k]] * nllbr[7] + f17[nv2[k]] * nlltr[7];	******	
ff18[k] =	f18[k] * nllbr[8] + f18[nv3[k]] * nllbl[8] + f18[nv2[k]] * nlltr[8] + f18[nv6[k]] * nlttl[8];	•	
ff20[k] =	: f20[k];		
ff21[k] =	f21[k] * nl2tr[] + f21[nv3[k]] * nl2tl[] + f21[nv7[k]] * nl2br[];		
ff22[k] =	f22[k] * nl2tl[2] + f22[nv4[k]] * nl2bl[2] + f22[nv4[k]] * nl2br[2];		
[£23[k] =	f23[k] * nl2bl[3] + f23[nv1[k]] * nl2br[3] + f23[nv2[k]] * nl2tr[3];		
ff24[k] =	. f24[k] * nl2br[4] + f24[nv3[k]] * nl2bl[4] + f24[nv2[k]] * nl2tr[4] + f24[nv6[k]] * nl2tl[4];		
ff25[k] =	f25[k] * nl2tr[5] + f25[nv3[k]] * nl2tl[5] + f25[nv7[k]] * nl2br[5];		
ff26[k] =	<pre>f26(k) * nl2tl(6) + f26(nv4(k)) * nl2bl(6) + f26(nv1(k)) * nl2tr(6) + f26(nv8(k)) * nl2br(6);</pre>		
4622 [4]	. f27[k] * nl2b1[7] + f27[nv1[k]] * nl2br[7] +		

Jun 20 1999 18:01	wet9is.c Page	age 32
£27	E27[nv2[k]] * nl2t1[7] + £27[nv5[k]] * nl2tx[7];	
u 77	f28[k] * nl2br[8] + f28[nv3[k]] * nl2b1[8] + f28[nv2[k]] * nl2tr[8] +	
break;		
case 1: ff10[k] = f1	£10[k];	
ff11[k] = f11	<pre>fil(k) * nlblbr[1] + f11[nv3[k]] * nlblb1[1] + f11[nv6[k]] * nlblt1[1] + f11[nv2[k]] * nlbltr[1];</pre>	
ff12[k] = f12	f12[k] * nlb1b1[2] + f12[nv1[k]] * nlb1br[2] + f12[nv2[k]] * nlb1t1[2] + f12[nv5[k]] * nlb1tr[2];	
££13[k] = £13	f13[k] * nl1b1[3] + f13[nv1[k]] * nl1br[3] + f13[nv2[k]] * nl1tr[3] +	
££14[k] = £14	f14[k] * nllbr[4] + f14[nv3[k]] * nllbl[4] + f14[nv2[k]] * nlltr[4] + f14[nv6[k]] * nlltl[4];	
ff15[k] = f18	<pre>f15[k] * nlblbl[5] + f15[nv1[k]] * nlblbr[5] + f15[nv2[k]] * nlbltl[5] + f15[nv5[k]] * nlbltr[5];</pre>	
ff16[k] = f1(<pre>f16[k] * nlblbr[6] + f16[nv3[k]] * nlblb1[6] + f16[nv2[k]] * nlbltr[6] + f16[nv6[k]] * nlblt1[6];</pre>	
££17[k] = £1;	<pre>f17[k] * nlib](7] + f17[nv1[k]] * nlibz[7] + f17[nv2[k]] * nlit1[7] + f17[nv5[k]] * nlitz[7];</pre>	
ff18[k] = f18	f18[k] * nllbr[8] + f18[nv3[k]] * nllbl[8] + f18[nv2[k]] * nlltr[8] + f18[nv6[k]] * nlltl[8];	
ff20[k] = f3	= f20[k];	
ff21[k] = f2:	<pre>£21[k] * nlb2br[i] + £21[nv3[k]] * nlb2bl[i] + £21[nv6[k]] * nlb2tl[i] + £21[nv2[k]] * nlb2tr[i];</pre>	
ff22[k] = f22	f22[k] * nlb2b1[2] + f22[nv1[k]] * nlb2br[2] + f22[nv2[k]] * nlb2t1[2];	
ff23[k] =	f23[k] * nl2b1[3] + f23[nv1[k]] * nl2br[3] + f23[nv2[k]] * nl2tr[3];	
ff24[k] = f2	<pre>£24[k] * nl2br[4] + £24[nv3[k]] * nl2bl[4] + £24[nv2[k]] * nl2tr[4] + £24[nv6[k]] * nl2tl[4];</pre>	
ff25[k] = f2:	f25[k] * nlb2b1[5] + f25[nv1[k]] * nlb2bx[5] + f25[nv2[k]] * nlb2tx[5];	
[[26[k] = [2]	<pre>f26(k) * nlb2br[6] + f26[nv3[k]] * nlb2bl[6] + f26[nv2[k]] * nlb2tr[6] + f26[nv6[k]] * nlb2tl[6];</pre>	
ff27[k] = f2	<pre>£27[k] * nl2b1[7] + £27[nv1[k]] * nl2br[7] + £27[nv2[k]] * nl2t1[7] + £27[nv5[k]] * nl2tr[7];</pre>	
ff28[k] = f2 break;	f28[hv2[k]] * nl2br[8] + f28[nv3[k]] * nl2bl[8] + f28[nv6[k]] * nl2tl[8];	
case 2: ff10[k] = f	f10[k];	
££11[k] = £1	fil(k) * nltr(1) + fil(nv3(k)) * nltr[1) + fil(nv7(k)) * nlb1(1) + fil(nv4(k)) * nlbr[1);	
ff12(k) = f1	<pre>f12[k] * nlt1[2] + f12[nv4[k]] * nl1b1[2] + f12[nv1[k]] * nltr[2] + f12[nv8[k]] * nl1br[2];</pre>	

_											<u>.</u>				<u> </u>										
Page 33										,															
Jun 20 1999 18:01 Wet9is.c	<pre>ffl3(k) = f13(k) * nltltl(3) + f13(nv1(k)) * nltltr[3] + f13(nv4(k)) * nltlbl(3) + f13(nv8(k)) * nltlbr(3);</pre>	<pre>ff14[k] = f14[k] * nltlr[4] + f14[nv3[k]] * nltlt[4] + f14[nv4[k]] * nltlbr[4] + f14[nv7[k]] * nltlbl[4];</pre>	<pre>ff15[k] = f15[k] * nlltr[5] + f15[nv3[k]] * nlltl[5] + f15[nv7[k]] * nllb1[5] + f15[nv4[k]] * nllbr[5];</pre>	<pre>ff16[k] = f16[k] * nl1t1[6] + f16[nv4[k]] * nl1b1[6] + f16[nv1[k]] * nl1tr[6] + f16[nv8[k]] * nl1br[6];</pre>	<pre>ffl7[k] = fl7[k] * nltltr[7] + fl7[nv3[k]] * nltltl[7] + fl7[nv4[k]] * nltlbr[7] + fl7[nv7[k]] * nltlbl[7];</pre>	<pre>ff18[k] = f18[k] * nlt11[8] + f18[nv1[k]] * nlt1x[8] + f18[nv4[k]] * nlt1b1[8] + f18[nv8[k]] * nlt1bx[8];</pre>	ff20[k] = f20[k];	<pre>ff21(k) = f21(k) * nl2tr[1] + f21[nv3(k)] * nl2tr[1] + f21[nv7(k]] * nl2tr[1] + f21[nv4[k]] * nl2tr[1];</pre>	<pre>ff22[k] = f22[k] * nl2tl[2] + f22[nv4[k]] * nl2bl[2] + f22[nv1[k]] * nl2tr[2] + f22[nv8[k]] * nl2br[2];</pre>	<pre>f£23(k) = f23(k) * nlt2tl(3) + f23(nv1(k)) * nlt2tr(3) + f23(nv4(k)) * nlt2bl(3) + f23(nv8(k)) * nlt2br(3);</pre>	<pre>ff24[k] = f24[k] * nlt2tr[4] + f24[nv3[k]] * nlt2tl[4] + f24[nv4[k]] * nlt2br[4] + f24[nv7[k]] * nlt2bl[4];</pre>	<pre>ff25[k] = f25[k] * nl2tx[5] + f25[nv3[k]] * nl2tx[5] + f25[nv7[k]] * nl2bx[5] + f25[nv4[k]] * nl2bx[5];</pre>	<pre>ff26[k] = f26[k] * nl2tl[6] + f26[nv4[k]] * nl2bl[6] + f26[nv1[k]] * nl2tr[6] + f26[nv8[k]] * nl2br[6];</pre>	<pre>ff27[k] = f27[k] * nlt2tr[7] + f27[nv3[k]] * nlt2tl[7] + f27[nv4[k]] * nlt2br[7] + f27[nv7[k]] * nlt2bl[7];</pre>	<pre>ff28[k] = f28[k] * nlt2tl[8] + f28[nv1[k]) * nlt2tr[8] + f28[nv4[k]] * nlt2bl[8] + f28[nv8[k]) * nlt2br[8];</pre>	break;	<pre>case 3: ff10[k] = f10[k];</pre>	<pre>ffl[k] = fll[k] * nlltl[l] + fll[nvl[k]] * nllltr[l] + fll[nv4[k]] * nlllb1[l] + fll[nv8[k]] * nlllbr[l];</pre>	<pre>ff12[k] = f12[k] * nlttl[2] + f12[nv4[k]] * nl1b1[2] + f12[nv1[k]] * nlttr[2] + f12[nv8[k]] * nlbr[2];</pre>	<pre>ff13[k] = f13[k] * n11b1[3] + f13[nv1[k]] * n11br[3] + f13[nv2[k]] * n11t1[3] + f13[nv5[k]] * n1tr[3];</pre>	<pre>ffl4[k] = fl4[k] * nlllbl(4] + fl4[nv1[k]] * nlllbr[4] + fl4[nv2[k]] * nllltl[4] + fl4[nv5[k]] * nllltr[4];</pre>	<pre>ff15[k] = f15[k] * nll1bl[5] + f15[nv1[k]] * nll1bx[5] + f15[nv2[k]] * nll1t1[5] + f15[nv5[k]] * nll1tx[5];</pre>	<pre>file[k] = fle[k] * nlltl[6] + fle[nv4[k]] * nllbl[6] + fle[nv1[k]] * nlltr[6] + fle[nv8[k]] * nllbr[6];</pre>	<pre>ffl7[k] = f17[k] * nllb1[7] + f17[nv1[k]] * nllbr[7] + f17[nv2[k]] * nllt1[7] + f17[nv5[k]] * nlltr[7];</pre>	

Jun 20 1999 18:01	wet9is.c	Page 34
££181k}	= £18[k] * nllltl[8] + £18[nv1[k]] *	
	f18[nv4[k]] * nlllbl[8] + f18[nv8[k]] * nlllbr[8];	
££20[k]	1 = £20(k);	
££21[k]	<pre>f21[k] * nll2tl[1] + f21[nv1[k]] * nll2tr[1] + f21[nv4[k]] * nll2b1[1] + f21[nv8[k]] * nll2br[1];</pre>	
ff22[k]	<pre> = f22[k] * nl2t1[2] + f22[nv4[k]] * nl2b1[2] + f22[nv1[k]] * nl2tr[2] + f22[nv8[k]] * nl2br[2];</pre>	
ff23[k]	<pre> = f23[k] * nl2b1[3] + f23[nv1[k]] * nl2br[3] + f23[nv2[k]] * nl2tr[3] + f23[nv5[k]] * nl2tr[3];</pre>	
ff24[k]] = £24[k] * nll2bl[4] + £24[nv1[k]] * nll2br[4] +	
££25[k]	<pre> = £25[k] * nll2bl[5] + £25[nvl[k]] * nll2br[5] + £25[nv2[k]] * nll2tl[5] + £25[nv5[k]] * nll2tr[5];</pre>	
££26[k]] = f26[k] * nl2tl[6] + f26[nv4[k]] * nl2bl[6] + f26[nv8[k]] * nl2br[6];	
££27[k]] = £27[k] * nl2bl[7] + £27[nv1[k]] * nl2br[7] + £27[nv2[k]] * nl2tl[7] + £27[nv5[k]] * nl2tr[7];	
ff28[k]] = £28[k] * nll2tl[8] + £28[nv1[k]] * nll2tr[8] + £28[nv4[k]] * nll2bl[8] + £28[nv8[k]] * nll2br[8];	
break;		
case 4: ff10[k]	:] = £10[k];	
ff11[k]] = f11[k] * nltr[l] + f11[nv3[k]] * nltl[l] + f11[nv7[k]] * nllb1[l] + f11[nv4[k]] * nllbr[l];	
ff12[k]	:] = f12[k] * nlritr[2] + f12[nv3[k]] * nlrit1[2] + f12[nv4[k]] * nlrib1[2];	
££13[k]	:] =	
ff14[k]	:] =	
££15[k]	:) = f15[k] * nlltr[5] + f15[nv3[k]] * nlltl[5] + f15[nv7[k]] * nllbr[5] + f15[nv4[k]] * nllbr[5];	
ff16[k]	:) = f16[k] * nlrlbr[6] + f16[nv3[k]] * nlrlb1[6] + f16[nv2[k]] * nlrlt1[6];	
££17 [k]	[] = f17(k] * nlrltr[7] + f17[nv3[k]] * nlrltl[7] + f17[nv4[k]] * nlrlbr[7];	
ff18[k]	:] = f18[k] * nllbr[8] + f18[nv3[k]] * nllbl[8] + f18[nv2[k]] * nlltr[8] + f18[nv6[k]] * nlltl[8];	
££20[k]	$c_1 = f20[k];$	
ff21[k]	<pre>c] = f21[k] * n12tx[1] + f21[nv3[k]] * n12t1[1] + f21[nv7[k]] * n12b1[1] + f21[nv4[k]] * n12bx[1];</pre>	
££22[k]	<pre>c) = f22(k] * nlr2tr[2] + f22(nv3(k]) * nlr2tl[2] + f22(nv4(k)] * nlr2br[2] + f22(nv7(k)) * nlr2bl[2];</pre>	
ff23[k]	<pre>c] = f23[k] * nlr2br[3] + f23[nv3[k]] * nlr2bl[3] + f23[nv2[k]] * nlr2tr[3] + f23[nv6[k]] * nlr2tl[3];</pre>	

91

62

Jun 20 1999 18:01	wet9is.c	Page 35
ff24[k] = f24[k] * f24[nv2[k]] *	nl2br[4] + £24[nv3[k]] * nl2bl[4] + nl2tr[4] + £24[nv6[k]] * nl2tl[4];	
ff25[k] = f25[k] * f25[nv7[k]] *	nl2tr[5] + f25[nv3[k]] * nl2tl[5] + nl2bl[5] + f25[nv4[k]] * nl2br[5];	
ff26[k] = f26[k] * f26[nv2[k]] *	nlr2br[6] + f26[nv3[k]] * nlr2bl[6] + nlr2tr[6] + f26[nv6[k]] * nlr2tl[6];	
ff27[k] = f27[k] * f27[nv4[k]] *	nlr2tr[7] + f27[nv3[k]] * nlr2tl[7] + nlr2br[7] + f27[nv7[k]] * nlr2bl[7];	
ff28[k] = f28[k] * f28[nv2[k]] * break;	nl2br[8] + £28[nv3[k]] * nl2bl[8] + nl2tr[8] + £28[nv6[k]] * nl2tl[8];	
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		

p.c	
wet9u	
2	
,	
,	
14:01	
1 1999 1	
Jun 11	

Jun 11 1999 14:01	wet9up.c	Page 1	2
/*************************************	**************************************		
<pre>#include <stdio.h> #include <stdib.h> #include <math.h> #include "wet9head.h"</math.h></stdib.h></stdio.h></pre>			
ouble co	boulk (int sigma, int index, int k, int nv, double nf10[], double nf12[], double nf12[], double nf15[], double nf15[], double nf16[], double nf16[], double nf21[], double nf22[], double nf21[], double nf22[], double nf24[], double nf24[], double nf24[], double nf24[], double nf25[],		
double dummy, ux, uy, uu, double gradx, grady; double fs1[9], fs2[9]; switch(sigma)	neq, cfl, source;		
<pre>case 1: cfl = cfl; break; case 2: cfl = cfl2; break; }</pre>			
fal(0) = nfl0 k - cfl fal(1) = nfl1 k - cfl fal(1) = nfl1 k - cfl fal(1) = nfl1 k - cfl k fal(2) = nfl2 k - cfl k fal(2) = nfl2 k - cfl k fal(2) = nfl5 k fal(2)	(nf10[k] - nf10[nv]); (nf11k] - nf11[nv]); (nf12[k] - nf12[nv]); (nf14[k] - nf13[nv]); (nf14[k] - nf15[nv]); (nf16[k] - nf15[nv]); (nf16[k] - nf15[nv]); (nf16[k] - nf16[nv]); (nf16[k] - nf18[nv]); (nf18[k] - nf20[nv]); (nf21[k] - nf21[nv]); (nf21[k] - nf21[nv]); (nf23[k] - nf23[nv]); (nf25[k] - nf25[nv]); (nf25[k] - nf25[nv]); (nf26[k] - nf26[nv]); (nf26[k] - nf26[nv]); (nf28[k] - nf28[nv]);	1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1	
Si	<pre>y_mode[k]) * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[4]+fs1[5]+</pre>	V-101N-1-	
* +	<pre>(fs1[1]*ecy1[1]+fs1[2]*ecy1[2]+fs1[3]*ecy1[3]+fs1[4]*ecy1[4]+ fs1[5]*ecy1[5]+fs1[6]*ecy1[6]+fs1[7]*ecy1[7]+fs1[8]*ecy1[8])</pre>		

Jun 11 1999 14:01	wet9up.c	Page 2
mass2 * (fs2[1]*ecy2[1]+ fs2[5]*ecy2[5]+ /tau2) / dummy;	(fs2[1]*ecy2[1]+fs2[2]*ecy2[2]+fs2[3]*ecy2[3]+fs2[4]*ecy2[4]+ fs2[5]*ecy2[5]+fs2[6]*ecy2[6]+fs2[7]*ecy2[7]+fs2[8]*ecy2[8]) dummy;	3)
<pre>break; case 1: ux = uxwall_bot; uy = uywall_bot; break; case 2: ux = uxwall_top; uy = uywall_top; break;</pre>		
<pre>case 3: ux = uxvall_left; uy = uyvall_left; break; case 4: ux = uxvall_right; uy = uyvall_right; break;</pre>		
<pre>uu = ux*ux + uy*uy; switch(sigma)</pre>		
case 1: dummy = (ecx1[index]*ux + /* printf("dummy=%25.20e\n",d	<pre>ecy1[index]*uy) / cspeed12; lummy);</pre>	
", " (fal[0]) + " (fal[0]) + " (fal[0]) + " (1.000 + three * dummy + three_over.two * uu / (fkey_init < 3)	<pre>"" "" "" "" "" "" "" "" "" "" "" "" ""</pre>	
<pre></pre>	* (fsl[index] - neq) + - csforce_x * ux - csforce_y * uy) * neq;	
gradx = gradn2x[k] - c grady = gradn2y[k] - c source = - ctaul * ((ecprod1[index] - c gforce * gradx *	<pre>Zx[k] - cfl * (gradnZx[k] - gradnZx[nv]); Zy[k] - cfl * (gradnZy[k] - gradnZy[nv]); taul * (fsl[index] - neq) + ndex] - csforce_x * ux - csforce_y * uy + grady * (ccxl[index] - ux) + grady * (ccxl[index] - uy)) * neq;</pre>	
/* source = - ctaul * (i (nloc2[K] - nloc2[K] (cspeed1 - ecx[index */	<pre>ctaul * (fs1[index] - neq) + kforce * neq * - nloo2[nv]) * - ecx[index] * ux - ecy[index] * uy);</pre>	
break; case 2: dunmy = (ecx2[index] * ux neq = ww[index] * (fs2[0]+	<pre>brak; lse 2: dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22; neq = ww[index] * (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+ (1.000 + three * dummy + nine_over_two * dummy * dummy - three_over_two * uu / cspeed22); if(key_init < 3)</pre>	
source = - ctau2 * ((fs2[index] - neq) + csforce_x * uy) * neq;	
{ gradx = gradnlx[k] - c grady = gradnly[k] - c	<pre>- ofl * (gradulx[k] - gradulx[nv]); - ofl * (graduly[k] - graduly[nv]);</pre>	

Page 4

Jun 11 1999 14:01

Jun 11 1999 14:01 Wet9up.c Page 3
<pre>source = - ctau2 * (fs2[index] - neq) + (ecprod2[index] - csforce_x * ux - csforce_y * uy + gforce * (gradx * (ecx2[index] - ux) +</pre>
/* / cource = - ctau2 * (fs2[index] - neq) + kforce * neg * (nloc1[k] - nloc1[nv]) * (cspeed2 - ecx[index] * ux - ecy[index] * uy); */
/* printf("iter=%d k=%d gradx=%e grady=%e source=%e gf=%e\n", iter,k,gradx,grady,source, fforce* (gradx * (ecx2[index]-ux) + grady * (ecy2[index]-uy))); */
} break;
return source;
<pre>double compute_upwind_sources_boundary(int sigma, int index, int k, int nv,</pre>
<pre>double dummy, ux, uy, uu, neq, cfl, source; double gradx, grady; double fsl[9], fs2[9];</pre>
switch(sigma)
case 1: cfl cfl cfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk); fsl cl nfl cfl (nfl ln nfl lk);
case 2: case 2: cfl2; cf
<pre>ux = ux_boundary - cfl * (uxloc[nv] - ux_boundary); uy = uy_boundary - cfl * (uyloc[nv] - uy_boundary);</pre>
<pre>uu = ux*ux + uy*uy; switch(sigma)</pre>
<pre>case 1: dummy = (ecx1[index]*ux + ecy1[index]*uy) / cspeed12;</pre>

The state of the s
<pre>neq = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[4]+fs1[5]+fs1[6]+</pre>
<pre>{ source = - ctaul * (fsl[index] - neq) +</pre>
<pre>f gradx = gradn2x[k] - cf1 * (gradn2x[k] - gradn2x[nv]); grady = gradn2y[k] - cf1 * (gradn2y[k] - gradn2y[nv]); source = - craul * (fs1[index] - neq) +</pre>
<pre>//cource = - ctaul * (fsl[index] - neq) + kforce * neq * (nloc2[k] - nloc2[nv]) * (cspeedl - ecx[index] * ux - ecy[index] * uy); */</pre>
brak; case 2; case 2: dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22; neq = ww[index] * (fs2[0]+fs2[1]+fs2[3]+fs2[4]+fs2[4]+fs2[5]+fs2[6]+ fs2[1]+fs2[8] * (fs2[8]) *
<pre>source = - ctau2 * (fs2[index] - neq) +</pre>
<pre>fgradx = gradnlx[k] - cfl * (gradnlx[k] - gradnlx[nv]); grady = gradnly[k] - cfl * (gradnly[k] - gradnly[nv]); source = - ctau2 * (fs2findex] - neq) +</pre>
/* /varce = - ctau2 * (fs2[index] - neq) + kforce * neq * (nloc1[k] - nloc1[nv]) * (cspeed2 - ecx[index] * ux - ecy[index] * uy); */
break; break; return source;
<pre>double compute_upwind_sources_boundary_in(int sigma, int index, int k, int nv,</pre>
<pre>double dummy, ux, uy, uu, neq, cfl, source; double gradx, grady; double fsl[9], fs2[9];</pre>
switch(sigma)
case 1; cfl = cfl1;

Jun 11 1999 14:01 Wet9up.c
<pre>fsl(0) = (nfl0[k] - cfl * (nfl0[nv] - nfl0[k])); fsl(1) = (nfl1[k] - cfl * (nfl1[nv] - nfl1[k])); fsl(2) = (nfl2[k] - cfl * (nfl2[nv] - nfl2[k])); fsl(3) = (nfl3[k] - cfl * (nfl3[nv] - nfl3[k])); fsl(4) = (nfl3[k] - cfl * (nfl4[k] - nfl4[k])); fsl(5) = (nfl5[k] - cfl * (nfl4[nv] - nfl4[k])); fsl(5) = (nfl5[k] - cfl * (nfl5[nv] - nfl5[k])); fsl(7) = (nfl5[k] - cfl * (nfl5[nv] - nfl5[k])); fsl(7) = (nfl8[k] - cfl * (nfl5[nv] - nfl8[k]));</pre>
/* printf("k=%d sigma=%d index-%d fs1=%e nf21[nv]=%e\n", k,sigma,index,fs1[1],nf11[k],nf11[nv]); */
Drask; Case 2: of1 = cf12; of2 = cf2; of3 = nf20[k] - cf1 * (nf20[nv] - nf20[k]); fs2[1] = nf21[k] - cf1 * (nf21[nv] - nf21[k]); fs2[2] = nf22[k] - cf1 * (nf22[nv] - nf22[k]); fs2[3] = nf22[k] - cf1 * (nf22[nv] - nf22[k]); fs2[4] = nf24[k] - cf1 * (nf24[nv] - nf24[k]); fs2[5] = nf25[k] - cf1 * (nf24[nv] - nf24[k]); fs2[6] = nf26[k] - cf1 * (nf25[nv] - nf25[k]); fs2[7] = nf27[k] - cf1 * (nf25[nv] - nf25[k]); fs2[8] = nf28[k] - cf1 * (nf28[nv] - nf28[k]);
<pre>// rintf("k=%d sigma=%d index=%d fs2=%e nf22[nv]=%e\n",</pre>
<pre>y - cfl * (uxloc[nv] - ux_boundary); y - cfl * (uyloc[nv] - uy_boundary);</pre>
<pre>printf("ite="#d k=#d index=#d ux=#e ux_boundary=#e uxloc[nv]=#e\n", iter,k,index,ux,ux_boundary,uxloc[nv]); vu = ux*ux + uy*uy; switch(sigma)</pre>
<pre>case 1: dummy : (cal[lindex]*ux + ecyl[index]*uy) / capeed12; dummy = ww[index] * (fs1[0]+fs1[1]+fs1[2]+fs1[3]+fs1[5]+fs1[6]+</pre>
<pre>source = - ctaul * (fsl[index] - neq) +</pre>
<pre>gradx = gradn2x[k] - cfl * (gradn2x[k] - gradn2x[nv]); grady = gradn2y[k] - cfl * (gradn2y[k] - gradn2y[nv]); source = - ctaul * (fsl[index] - neg) +</pre>
<pre>source = - ctaul * (fsl[index] - neq) + kforce * neq * (nloc2[k] - nloc2[nv]) * (cspeedl - ecx[index] * ux - ecy[index] * uy); */</pre>
; (neq < 0)

Jun 11 1999 14:01 Wet9up.c Pa	Page 6
<pre>printf("negative neg sumfsl=#e term=#e\n",</pre>	
<pre>printf("iter=%d k=%d index=%d fs1=%25.20e neq=%25.20e source=%25.20e =%25.20e fs1+source=%25.20e ux=%e uy=%e ecprod=%e\n", iter,k,index,fs1[index],neq,source,fs1[index]-neq,fs1[index]+source,ux,uy,e</pre>	fsl-neq ux, uy, e
= - ctau2 * (fs2[index] - rod2[index] - csforce_x * un gradn1x[k] - cfl * (gradn1) gradn1y[k] - cfl * (gradn1)	
<pre>b = _ crau2 * (fs2[index] - neq) cprod2[index] - csforce_x * ux - cs force * (gradx * ux + grady * uy)) b = _ crau2 * (fs2[index] - neq) col[k] - nloc1[nv]) * peed2 - ecx[index] * ux - ecy[index]</pre>	
<pre>intf("iter=%d k=%d index=%d fs2=%25.20e neq=%25.20e fs2+source=%25.20e ux=%e uy=%e ecprod=%e\n", iter,k,index,fs2[index],neq,source,fs2[index]-neq,fs2[index]+source, ndex]); (neq < 0)</pre>	fs2-neq ux,uy,e
<pre>printf("Inggalive neq sumfs2=%e term=%e\n",</pre>	

Jun 11 1999 14:01 Page 7	Jun 11 1999 14:01
<pre>printf("fsl=%e nfil=%e nf[nv]=%e cfl=%e\n", printf("fsl=%e nfil=%e nfil=%e cfl=%e\n", printf("fs2=%e nfilv)=%e cfl=%e\n", fs1[2].nf12[k].nf12[nv].cfl); printf("fs3=%e nfil=%e nfil)=%e cfl=%e\n", printf("fs3=%e nfil=%e nfin)=%e cfl=%e\n", printf("fs3=%e nfil=%e nfin)=%e cfl=%e\n", printf("fs5=%e nfil=%e nfin)=%e cfl=%e\n", printf("fs5=%e nfil=%e nfin)=%e cfl=%e\n", printf("fs5=%e nfil=%e nfin)=%e cfl=%e\n", fs1[6].nf15[k].nf15[nv]-cfl); printf("fs6=%e nfin)=%e cfl=%e\n", fs1[6].nf16[k].nf16[nv]-%e cfl=%e\n", fs1[7].nf17[k].nf17[nv]-%e cfl=%e\n", fs1[8].nf17[k].nf17[nv]-%e cfl=%e\n", fs1[8].nf18=%e nfin)=%e cfl=%e\n", fs1[8].nf18=%e nfin)=%e\n", fs1[8].nf18=%e nfin)=%e\n", fs1[8].nf18=%e\n", fs1[8].nf18=%e\n", fs1[8].nf18=%e\n", fs1[8].nf18=%e\n", fs1[8].nf18=%e\n", fs1[8].nf18=%e\n", fs1[8].nf18=%e\n",</pre>	(case 1: dummy = (ecxl dummy = (ecxl dumes = ww[index) fil (Not + three_over, three_over, if (key_init <
// break;	(eoprod y* source = - (nloc2)
<pre>double compute_upwind_sources_boundary_out(int sigma, int index, int nv, int k,</pre>	(cspeed)
<pre>double dummy, ux, uy, uu, neq, cfl, source; double gradx, grady; double fal[9], fa2[9]; switch(sigma)</pre>	if(key_init <
cfl = cfl1; cfl = cfl1; fsl[0] = nfl0[k] + cfl * (nfl0[nv] - nfl0[k]); fsl[1] = nfl1[k] + cfl * (nfl1[nv] - nfl1[k]); fsl[2] = nfl2[k] + cfl * (nfl2[nv] - nfl2[k]); fsl[3] = nfl3[k] + cfl * (nfl2[nv] - nfl2[k]); fsl[4] = nfl4[k] + cfl * (nfl4[nv] - nfl4[k]); fsl[5] = nfl5[k] + cfl * (nfl5[nv] - nfl4[k]); fsl[6] = nfl5[k] + cfl * (nfl5[nv] - nfl4[k]); fsl[6] = nfl5[k] + cfl * (nfl5[nv] - nfl7[k]); fsl[8] = nfl8[k] + cfl * (nfl8[nv] - nfl7[k]);	gradx = gr grady = gr grady = gr source = (ecproc /* gforce /* (cspeedd. */
of12; = nf20[k] + cfl * (nf20[nv] - = nf21[k] + cfl * (nf21[nv] -	break; } return source; }
nf22[k] + cfl * (nf22[nv] nf23[k] + cfl * (nf22[nv] nf24[k] + cfl * (nf24[nv] nf25[k] + cfl * (nf25[nv] nf25[k] + cfl * (nf25[nv] nf27[k] + cfl * (nf26[nv] nf27[k] + cfl * (nf28[nv] nf28[k] + cfl * (nf28[nv]	void compute_upwind.
<pre>ux = ux_boundary + cfl * (uxloc[nv] - ux_boundary); uy = uy_boundary + cfl * (uyloc[nv] - uy_boundary); uu = ux*ux + uy*uy; switch(sigma)</pre>	for (k=0; k <nnodes. =="" nf2)<="" sf10[k]="nf1(sf20[k]" td="" {=""></nnodes.>

x + ecyl[index]*uy) / c [0]+fsl[1]+fsl[2]+fsl[3] [8], white_over_two * du my + nine_over_two * du cspeedl2); * (fsl[index] - neq) - csforce_x * ux - csf - cfl * (dradn2x[k] -
= grading k - cfl * (grading k - cef prote_x * ux - cef prote_x *
<pre>break; case 2: case 2: dummy = (ecx2[index] * ux + ecy2[index] * uy) / cspeed22; dummy = (fs2[0]+fs2[1]+fs2[2]+fs2[3]+fs2[4]+fs2[5]+fs2[6]+</pre>
<pre>source = - ctau2 * (fs2[index] - neq) +</pre>
<pre>gradx = gradnlx[k] - cfl * (gradnlx[k] - gradnlx[nv]); grady = gradnly[k] - cfl * (gradnly[k] - gradnly[nv]); gource = - cfau2 * (fs2[index] - neq) +</pre>
<pre>source = - ctau2 * (fs2[index] - neq) + kforce * neq * (nloc1[k] - nloc1[nv]) *</pre>
} return source;
<pre>void compute_upwind_sources(double nf10[], double nf12[],</pre>
int k;
<pre>for(k=0; k<nnodes_all; k++)="" pre="" {<=""></nnodes_all;></pre>
<pre>sf10[k] = nf10[k]; sf20[k] = nf20[k]; source10[k] = - ctaul * (nf10[k] - neq10[k]) -</pre>

Jun 11 1999 14:01

Page 9

wet9up.c

Jun 11 1999 14:01

Page 10

<pre>sf18[k] = nf18[k] - cf11 * (nf18[k] - nf18[nv6[k]]); source18[k] = compute_upwind_sources_bulk(1, 8, k, nv6[k],</pre>	<pre>sizi(k) = nizi(k) = ciri. (nizi(k) = nizi(n)(k)); y) source21[k] = compute_upwind_sources_bulk(2, 1, k, nv3[k],</pre>	<pre>sf22[k] = nf22[k] - cf12 * (nf22[nv2[k]] - nf22[k]); source22[k] = compute_upwind_sources_boundary_in(2, 2, k, nv2[k],</pre>	<pre>sf23[k] = nf23[k] - cf12 * (nf23, nf24, nf25, nf26, nf27, nf28); source23[k] = compute_upwind_nesources_bulk[k]]; source23[k] = compute_upwind_sources_bulk[k], r nv1[k], nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]); source14[k] = compute_upwind_sources_boundary_out(1, 4, nv2[k], k,</pre>	1 2	<pre>sf15[k] = nf15[k] - cf11 * (nf15[nv5[k]] - nf15[k]); sourcel5[k] = compute_upwind_sources_boundary_in(1, 5, k, nv5[k],</pre>	<pre>sfl6[k] = nfl6[k] - cfll * (nfl6[nv6[k]] - nfl6[k]); sourcel6[k] = compute_upwind_sources_boundary_in(1, 6, k, nv6[k],</pre>
<pre>(csforce_x * uxloc(k] + csforce_y * uyloc(k]) * neq10[k]; source20[k] = - ctau2 * (nf20[k] - neq20[k]) - (csforce_x * uxloc(k] + csforce_y * uyloc(k]) * neq20[k]; (csforce_x * uxloc(k] + csforce_y * uyloc(k]) * neq20[k]; switch(boundary_mode[k]) (case 0:</pre>	<pre>sfl2[k] = nf12[k] - cfl1 * (nf12[k] - nf12[nv4[k]]); source12[k] = compute_upwind_sources_bulk(l, 2, k nv4[k],</pre>	<pre>sf13[k] = nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]); source13[k] = compute_upwind_sources_bulk(1, 3, k, nv1[k],</pre>	<pre>sfl4[k] = nfl4[k] - cfl1 * (nfl4[k] - nfl4[nv2[k]]); sourcel4[k] = compute_upwind_sources_bulk(l, 4, k, nv2[k],</pre>	<pre>sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]); source15[k] = compute_upwind_sources_bulk(1, 5, k, nv7[k],</pre>	<pre>sf16[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]); source16[k] = compute_upwind_sources_bulk(1, 6, k, nv8[k],</pre>	<pre>sf17[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]); sourcel7[k] = compute_upwind_sources_bulk[1, 7, k, nv5[k],</pre>

wet9up.c

67

	sf25[k] source25 sf16[k] source1(sf26[k]
<pre>### ### ### ### ### ### ### ### ### ##</pre>	sourcell sourcell sourcell sourcell
	sf16[k] source1(sf26[k]
Doct, upwall_Doct, a_upwald_sources_boundary_out[1]; ffl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, ffl1 * (nfl8[k] - nfl8[nv6[k]]); bot, upwall_boc, ffl1, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, ffl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, ffl1, nfl2, nfl2, nfl8, nfl8, ffl1, nfl2, nfl2, nfl8, nfl8, ffl1, nfl2, nfl2, nfl8, ffl1, nfl2, nfl2, nfl8, ffl1, nfl8, ffl2,	sf26[k] source2
######################################	source2(K)
Jobo, Waralloof, 1813, nf14, nf15, nf16, nf17, nf18, nf12, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf22, nf23, nf24, nf25, nf26, nf27, nf28); ff21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); bot, uywall_bot, nf15, nf14, nf15, nf16, nf17, nf18, nf11, nf12, nf13, nf14, nf15, nf26, nf27, nf28); */ cf11 * (nf11[k] - nf11[nv3[k]]); selupaind_sources_bulk(l, l, k, nv3[k], nf18, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf11, nf12, nf11, nf12, nf11, nf15, nf16, nf17, nf18, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf11, nf12, nf12, nf12, nf25, nf26, nf27, nf28); cf11 * (nf12[k] - nf12[nv4[k]]); cf11 * (nf12[k] - nf12[nv4[k]]);	
nfll, nfl2, nfl3, nfl4, nfl5, nfl6, nfl7, nfl8, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); ff21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); ff1 * (nfll[k] - nfl1[nv3[k]]); e.upwind.sources_bulk(1, 1, k, nv3[k], ff1, nf2, nf13, nf14, nf15, nf16, nf17, nf18, ff21, nf23, nf24, nf25, nf26, nf27, nf28); ff21 * (nf2[k] - nf2[k] - nf2[k] + nf15, nf16, nf17, nf18, nf1, nf2, nf13, nf14, nf15, nf16, nf17, nf18, ff21, nf22[k] - nf12[nv4[k]); ef11 * (nf12[k] - nf12[nv4[k]);	sf17[k] sourcel
fil * (nfil[k] - nfil[nv3[k]]); supwind_sources_bulk(l, 1, k, nv3[k], supwind_sources_bulk(l, 1, k, nv3[k], nfil, nfi2, nfi3, nfi4, nfi4, nfi5, nfi6, nfi7, nfi8, nfi2, nfi2, nfi3, nfi4, nfi2, nfi2, nfi2, supwind_sources_bulk(2, 1, v, v3[k], nfi1, nfi2, nfi3, nfi4, nfi5, nfi6, nfi7, nfi8, nfi1, nfi2, nfi3, nfi4, nfi5, nfi6, nfi7, nfi8, nfi1, nfi2[k] - nfi2[nv4[k]]); cfi1 * (nfi2[k] - nfi2[nv4[k]);	sf27[k]
), nELY, nELS, nELS, nELS, nELS, nELS, nELS, nELS); - cfl2 * (nf21[K] - nf21[nv3[K]]); oute_upwind_sources_bulk(2, 1, K, nv3[K], nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); - cfl1 * (nf12[K] - nf12[nv4[K]]);	sourcez
- cfll * (nfl2[k] - nfl2[nv4[k]]);	sf18[k] source1
ubte_upwind_sources_boundary_out(1, 2, nv4[k], k,	sf28[k] source2
######################################	break; case 3: sf11[k] source1
<pre>= nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]); lk] = compute_upwind_sources_bulk(1, 3, k, nv1[k], nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf24, nf25, nf26, nf27, nf28);</pre>	sf21(k) source2
nf18, nf28);	sf12[k]
<pre>sfl4(k] = nf14(k) - cfl1 * (nf14(nv4(k)) - nf14(k)); sourcel4(k) = compute_upwind_sources_boundary_in(1, 4, k, nv4(k),</pre>	source. source2
	sf13[k] source1
<pre>sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k])); source15[k] = compute_upwind_sources_boundary_out(1, 5, nv7[k], k,</pre>	sf23[k]

Jun 11 1999	14:01 wet9up.c Page 1
	<pre>uxwall_top, uywall_top, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28); source25[k] = nf25[k] - nf25[k] - nf25[k]/[k]]); source25[k] = compute_upwind_sources_boundary_out(2, 5, nv7[k], k, uxwall_top, uywall_top, uywall_top, nf12, nf13, nf14, nf18, nf10, nf11, nf12, nf13, nf14, nf15, nf26, nf27, nf28); nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);</pre>
ပ်() လို() ဝိ	<pre>sf16[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]); source16[k] = compute_upwind_sources_boundary_out(1, 6, nv8[k], k,</pre>
41 Ö	<pre>sel7[k] = nf17[k] - cf11 * (nf17[nv7[k]] - nf17[k]); source17[k] = compute_upwind_sources_boundary_in(1, 7, k, nv7[k],</pre>
41 O 41 O	<pre>sfl8(k) = nfl8(k) - cfl1 * (nfl8(nv8(k)] - nfl8(k)); sourcel8(k) = compute_upwind_sources_boundary_in(1, 8, k, nv8(k),</pre>
A C C C C C C C C C C C C C C C C C C C	<pre>eak; 11 [k] = nfil[k] - cfil * (nfil[nv1[k]] - nfil[k]); urcell[k] = compute_upwind_sources_boundary_in(1, 1, k, nv1[k],</pre>
	<pre>sel2[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]); source12[k] = compute_upwind_sources_bulk[l, 2, k, nv4[k],</pre>
es s	<pre>sf13[k] = nf13[k] - cf11 * (nf13[k] - nf13[nv1[k]]); sourcel3[k] = compute_upwind_sources_boundary_out(1, 3, nv1[k], k,</pre>

1999 14:01 Wet9up.c Page 13
<pre>ource23[k] = compute_upwind_sources_boundary_out(2, 3, nv1[k], k,</pre>
<pre>sfl4(k) = nf14(k) - cfl1 * (nf14(k) - nf14[nv2(k]]); source14(k) = compute_upwind_sources_bulk(l, 4, k, nv2(k),</pre>
<pre>set15[k] = nf15[k] - cf11 * (nf15[nv5[k]] - nf15[k]); source15[k] = compute_upwind_sources_boundary_in(1, 5, k, nv5[k],</pre>
<pre>sci6[k] = nf16[k] - cf11 * (nf16[k] - nf16[nv8[k]]); source16[k] = compute_upwind_sources_boundary_out(1, 6, nv8[k], k,</pre>
<pre>sel7[k] = nf17[k] - cf11 * (nf17[k] - nf17[nv5[k]]); source17[k] = compute_upwind_sources_boundary_out(1, 7, nv5[k], k, uxwall_left, uywall_left, lf14, nf15, nf16, nf17, nf18, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, sf27[k] = nf27[k] - cf12 * (nf27[k] - nf27[nv5[k]]); source27[k] = compute_upwind_sources_boundary_out(2, 7, nv5[k], k, nf10, nf11, nf12, nf13, nf14, nf15, nf16, nf17, nf18, nf20, nf21, nf22, nf23, nf24, nf25, nf26, nf27, nf28);</pre>
<pre>sel8[k] = nf18[k] - cf11 * (nf18[nv8[k]] - nf18[k]); source18[k] = compute_upwind_sources_boundary_in(1, 8, k, nv8[k],</pre>
<pre>break; case 4: case 4: sfl[k] = nfl[k] - cfl1 * (nfl1[k] - nfl1[nv3[k]]); sfl1[k] = compute_upwind_sources_boundary_out(l, 1, nv3[k], k,</pre>

Jun 11 16	999 14:01 wet9up.c	Page 14
	<pre>sf12[k] = nf12[k] - cf11 * (nf12[k] - nf12[nv4[k]]); source12[k] = compute_upwind_sources_bulk[i, 2, k, nv4[k],</pre>	
	<pre>sf13[k] = nf13[k] - cf11 * (nf13[nv3[k]] - nf13[k]); source13[k] = compute_upwind_sources_boundary_in(1, 3, k, nv3[k],</pre>	
	<pre>sf14[k] = nf14[k] - cf11 * (nf14[k] - nf14[nv2[k]]); source14[k] = compute_upwind_sources_bulk[1, 4, k, nv2[k],</pre>	
	<pre>sf15[k] = nf15[k] - cf11 * (nf15[k] - nf15[nv7[k]]); source15[k] = compute_upwind_sources_boundary_out(1, 5, nv7[k], k,</pre>	
	<pre>sfi6(k) = nfi6(k) - cfil * (nfi6(nv6(k)) - nfi6(k)); source16(k) = compute_upwind_sources_boundary_in(l, 6, k, nv6(k),</pre>	
	<pre>sf17(k] = nf17(k] - cf11 * (nf17(nv7(k)) - nf17(k)); source17(k) = compute_upwind_sources_boundary_in(1, 7, k, nv7(k),</pre>	
	<pre>sf18[k] = nf18[k] - cfl1 * (nf18[k] - nf18[nv6[k]]); sourcel8[k] = compute_upwind_sources_boundary_out(1, 8, nv6[k], k,</pre>	

ain.c	
wet9m	, in the second
10	7
Jun 27 1999 13:08	^
يال	,

Page 2

Jun 27 1999 13:05 Wet9main.c	Page 1
/*************************************	
<pre>#define MAIN_HEADER #include <stdio.h> #include <stdib.h> #include <math.h></math.h></stdib.h></stdio.h></pre>	
<pre>void getavec_square(void); void getavec_square(void); void wet9_input(void); void wet9_input(void); void dest_ntot(void); void test_ntot(void); void test_ntot(void); void test_ntot(void); void test_ntot(void); void test_ntot(void); void xv_new(double n0[], double n1[], double n5[], double n1[], double n2[], double n3[], double n4[], double n1[], double n5[], double n3[], void xv(double n0[], double n1[], double n5[], double n7[], void intt_arrays_nine_square(void); void intt_arrays_nine_square(void); void wet9_channel_profile(void); void wet9_channel_profile(void); void wet9_channel_profile(void);</pre>	
e nf10[]; e nf13[]; e nf20[]; e nf23[]; e nf23[];	
<pre>double dummy, uxloc1, uyloc2, uyloc2; for (k=0; k<nnodes_all; k++)<="" td=""><td></td></nnodes_all;></pre>	
<pre>switch (Doubledry) (case 0: case 1: case 2: case 2: case 2:</pre>	
uxloc2 = (nf21[k]*ecx2[1]+nf22[k]*ecx2[2]+nf23[k]*ecx2[3]+ nf24[k]*ecx2[3]+nf25[k]*ecx2[6]+nf26[k]*ecx2[6]+ nf27[k]*ecx2[7]+nf28[k]*ecx2[6]); uyloc2 = (nf21[k)*ecy2[1]+nf22[k]*ecy2[2]+nf23[k]*ecy2[3]+ nf24[k]*ecy2[1]+nf25[k]*ecy2[5]+nf23[k]*ecy2[6]+	

2
ag3.mec.utt
flumag3.
from
sofonea
Printed by
_

Jun 27 1999 13:05 Wet9main.c	Page 3	
<pre>dummy = (ecx1[6]*uxloc[k] + ecy1[6]*uyloc[k]) / cspeed12; neq16[k] = w2*nloc1[k] * (1.000 + three * dummy +</pre>		
<pre>dummy = (ecx2[1]*uxloc[k] * (1.000 - uu2); dummy = (ecx2[1]*uxloc[k] * (1.000 + three * dummy + dummy - uu2); neq21[k] = w1*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq22[k] = w1*nloc2[k] * (1.000 + three * dummy + dummy - uu2); dummy = (ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22; neq23[k] = w1*nloc2[k] * nine_over_two * dummy * dummy - uu2); dummy = (ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22; neq24[k] = w1*nloc2[k] * nine_over_two * dummy * dummy - uu2); neq24[k] = w1*nloc2[k] * nine_over_two * dummy * dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq26[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq26[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq26[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq26[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq27[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy * dummy - u</pre>		
<pre>printf("nl=%lf neql=%lf n</pre>		
<pre>int k; double dummy = 3.000 / delta_x; for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>		
<pre>{ switch(boundary_mode[k])</pre>		•

Jun 27 1999 13:05	wet9main.c Page 4
ecx[1]*nloc1[nv1[k]] ecx[3]*nloc1[nv3[k]]	1[k]] + ecx[2]*nloc1[nv2[k]] + 3[k]] + ecx[4]*nloc1[nv4[k]]) +
<pre>ecx[5] *nloc1[nv5[k]] ecx[7] *nloc1[nv7[k]] gradn2x[k] = dumny *</pre>	5[k]] + ecx[6]*nloc1[nv6[k]] + 7[k]] + ecx[8]*nloc1[nv8[k]]));
× ×	1[k]] + ecx[2]*nloc2[nv2[k]] + 3[k]] + ecx[4]*nloc2[nv4[k]]) +
<pre>w</pre>	<pre>75[k]] + ecx[6]*nloc2[nv6[k]] + 7[k]] + ecx[8]*nloc2[nv8[k]]);</pre>
	11[k]] + ecy[2]*nloc1[nv2[k]] + 13[k]] + ecy[4]*nloc1[nv4[k]]) +
= "cy[5] *nloc1[nv5[k]] ecy[7] *nloc1[nv7[k]] gradn2y[k] = dummy *	\frac{1}{1} + \text{6} \text{*nloc1} \text{nv6}[k]] + \text{7}[k] + \text{ecy}[8] \text{*nloc1} \text{nv8}[k]]);
	11[k]] + ecy[2]*nloc2[nv2[k]] + 73[k]] + ecy[4]*nloc2[nv4[k]]) +
W2 * (evg [5] *nloc2 [nv5 [K]] ecg [7] *nloc2 [nv7 [K]]	5[k]] + ecy[6]*nloc2[nv6[k]] + r7[k]] + ecy[8]*nloc2[nv8[k]]));
Dreak; case 2: break;	
void first_upwind(double mass, double te double nf0[], double r double nf3[], double n double nf6[], double nf0[], double n	<pre>louble tau, double cspeed, double nf1[], double nf2[], double nf4[], double nf8[], double nf7[], double nf8[], double nf7[], double nf8[],</pre>
nff6[], neq0[], neq3[], neq6[], ecprod[double nff4[], double nff5[], double nff7[], double nff8[], double neq1[], double neq2[], double neq4[], double neq8[], double sforce, double grady[]
int k; double ctau, cgrad, cgradx, cgrady, cg double cgradx, cgrady2, cgradx2; double cgradx2; double quamy_force; double prodscal, color_x, color_y, co.	ogradxs2, cgradys2, cgradxy; colormod, nn;
ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_y; cgradxs2 = cspeed * sgrt((double) 2) * delta_t / delta_t cgradxy = cspeed * sgrt((double) 2) * delta_t / delta_t cgradxy = cspeed * delta_t / sgrt((delta_x**delta_x*) + cgradxy = cspeed * delta_t / (2.000*delta_x*); cgradxy2 = cspeed * delta_t / (2.000*delta_x*); cgradxy2 = cspeed * delta_t / (2.000*delta_y);	<pre>* delta_t / delta_x; * delta_t / delta_y; ta_x*delta_x + delta_y*delta_y); elta_x); elta_y); a x*delta x + delta v*delta v);</pre>
for (k=0; k <nnodes_all; k++)<="" td=""><td></td></nnodes_all;>	

Jun 27 1999 13:05 wet9main.c	Page 5
<pre>f prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k]; switch(boundary_mode[k])</pre>	
<u> </u>	
- cgradx - ceX[s] * (ins]iNulfs] - n.clk]) + (cprod[3] - p.cdscal) * insg3[k]; (cprod[3] - p.cdscal) * insg3[k] * chrome n.ff4[k] - (inf4[k] - cf4[k]) * cf4[k] * c	
- cgrady * ccv[6] * (nf5[k] - nf5[mv4[k]]) + (cgrady * ccv[6]) * ncg[k]; k]; (cgrady * ccv[6]) * ncg[k]; k];	
<pre>nrio(k) = nrio(k) = (nrio(k) nrio(k))</pre>	
(ecprod(s) - productal) " nege(k);	
7] - prodscal) * neq7[k] nf8[k] - (nf8[k]-neq8[k] * ecx[8] * (nf8[k] - nf * ecx[8] * (nf8[ny2[k])	
8] - prodscal) * neq8[k]; terface)	
<pre>case 0: /* nothing */ break; case 2: /* color field gradient */ nn = nf0[k] + nf1[k] + nf2[k] + nf3[k] + nf5[k] + nf6[k] nf7[k] + nf8[k]</pre>	+
<pre>/* color_x = sforce * (colorfield[nv1[k]]-colorfield[nv3[k]]) /</pre>	
<pre>(2.000 * delta_x); color_y = sioree * (colorfield[nv2[k]]-colorfield[nv4[k]]) / color_y = sioree * (colorfield[nv4[k]]-colorfield[nv4[k]]) / color_y = sioree * (colorfield[nv4[k]]-colorfield[n</pre>	
= sforce delta_x	
sforce * nn * (colorfield[ndelta_x * mass);	_
color_y = sforce * nn * grady[k] / mass;	
olor_x = sforce * (2.000 * delta_x	
= neq0[k] *	ا آخ
_x = sforce * nn lta_x * mass);	
<pre>color_y = 0.000; prodecal = color_x * uxloc(k] + color_y * uyloc(k); nffl(k] += neq1(k] * (ecx[1]*cspeed*color_x + ecy[1]*cspeed*color_y</pre>	ו קל
ı	

Page 6			1 1	1 1
wet9main.c	<pre>prodscal); nn * (colorfield[k]-colorfield[nv4[k]]) / * uxloc[k] + color_y * uyloc[k]; * (exz[2]*cspeed*color_x + exy[2]*cspeed*color_y prodscal); nn * (colorfield[nv1[k]]-colorfield[k]) / * uxloc[k] + color_y * uyloc[k]; * uxloc[k] + color_y * uyloc[k]; * cox[3]*cspeed*color_x + exy[3]*cspeed*color_y redscal);</pre>	nn * (colorfield[k]-colorfield[nv4[k]]) / * uxloc[k] * (colorfield[k]-colorfield[nv4[k]]) / * (colorfield[k]-colorfield[nv3[k]]) / nn * (colorfield[k]-colorfield[nv4[k]]) / * uxloc[k] + color_y * uyloc[k]; * (cox[5]*cspeed*color_x + coy[5]*cspeed*color_y prodscal); nn * (colorfield[nv1[k]]-colorfield[k]) / nn * (colorfield[nv1[k]]-colorfield[k]) /	<pre>n * (colorfield[k]-colorfield[nv4[k]]) / uxloc[k] + color_y * uyloc[k]; (ccx[6]*cspeed*color_x + ccy[6]*cspeed*color_y prodscal); n * (colorfield[nv1[k]]-colorfield[k]) / n * (colorfield[nv2[k]]-colorfield[k]) / uxloc[k] + color_y * uyloc[k]; (ccx[7]*cspeed*color_x + ccy[7]*cspeed*color_y prodscal); n * (colorfield[k]-colorfield[nv3[k]]) / n * (colorfield[nv2[k]]-colorfield[k]) / uxloc[k] + color_y * uyloc[k]; (ccx[8]*cspeed*color_x + ccy[8]*cspeed*color_y prodscal);</pre>	<pre>field gradient without nn */ l[k] + n£2[k] + n£3[k] + n£4[k] + n£5[k] + nf6[k] + i, (colorfield[nv1[k]]-colorfield[nv3[k]]) /</pre>
Jún 27 1999 13:05	* \$ \$ TO * \$ \$ \$ TO	color_x = 0.000; color_y = sforce * r (delta_x * mass); prodscal = color_x * nff4[k] += neq4[k] * color_x = sforce * r (delta_x * mass); prodscal = color_x * nff5[k] += neq5[k] * nff5[k] += neq5[k] * color_x = sforce * r (delta_x * mass); prodscal = color_x * nff5[k] += neq5[k] *	sforce * * condas) * condas) * ned6[K] * sforce * * color_X * mass) * sforce * * mass) * sforce * * mass) * ned8[K]	case 4: /* color field nn = nf0[k] + nf1[k] nf7[k] + nf8[k]; color_x = sforce * (c. 0.00 * delta_x * (c. 0.00 * delta_x * (c. 0.00 * delta_x * nf0[k] += neq0[k] * color_x = sforce * (delta_x * mass); color_x = sforce * (delta_x * mass); color_x = sforce * (c. 0.00; color_x = 0.000;

Jun 27 1999 13:05

Jun 27 1999 13:05 Wet9main.c	7
<pre>prodscal = color.x * uxloc(k] + color.y * uyloc(k]; nff2(k] += neq2(k] * (ecx[2]*cspeed*color.x + ecy[2]*cspeed*color.y -</pre>	
= sforce * (co	
<pre>color_y = 0.000; prodscal = color_x * uxloc[k] + color_y * uyloc[k]; nff3[k] += neq3[k] * (ecx[3]*cspeed*color_x + ecy[3]*cspeed*color_y - prodscal);</pre>	
1.000; iforce * (co	
<pre>prodecal = color_x * uxloc[k] + color_y * uyloc[k]; nff4[k] += neq4[k] * (ext[4]sepeed*color_x + ecy[4]*cspeed*color_y -</pre>	
<pre>color_x = sforce * (colorfield[k]-colorfield[nv3[k]]) / (delta ** mass)</pre>	
	· · · · · · · · · · · · · · · · · · ·
<pre>prodscal); color_x = sforce * (colorfield[nv1[k]]-colorfield[k]) /</pre>	
44.	
= sforce *	
۲ű×	
color_x neq7[k]	
orce *	
**	
<pre>productal = color_x * uxloc[k] + color_y * uvloc[k]; nff8[k] += neq8[k] * (exc[speed*color_x + ecy[8]*cspeed*color_y -</pre>	
Dreak; case 3: /* HE */	
<pre>color_x = (colorfield[k]-colorfield[nv3[k]]) / delta_x; color_y = 0.000; colormed = sqrt(color_x*color_x + color_y*color_y); prodscal = (color_x * exr[l] + color_y * ecy[l]) * cspeed;</pre>	
<pre>1f(colormod) * iff[k += neq1 k * (kforce / (2.000 * colormod)) * (color_x*prodscal - colormod*ecx[1]*cspeed/2.000) * (color_y*prodscal - uxloc[k]) +</pre>	
<pre>color_x = 0.000; color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x; colormod = sqrt(color_x*color_x + color_y*color_y); proddcal = (color_x * ecx[2] + color_y * ecy[2]) * cspeed;</pre>	
<pre>1.(color_w*prodscal - colormod*ecx[2]*cspeed/2.000) * ((color_x*prodscal - colormod*ecx[2]*cspeed/2.000) * (ecx[2]*cspeed - uxloc[k]) + (color_y*prodscal - colormod*ecy[2]*cspeed/2.000) * (ecy[2]*cspeed - uyloc[k]);</pre>	
	7

	<pre>color.x = (colorfield[nv1[k]]-colorfield[k]) / delta_x; color.y = 0.000; color.nod = sqrt(color.x*color.x + color.y*color.y); prodscal = (color.x * ecx[3] + color.y * ecy[3]) * cspeed;</pre>
	<pre>if(colormod) ** if(color_x*prodscal - colormod*ecx[3]*cspeed/2.000) * (color_x*prodscal - colormod*ecx[3]*cspeed/2.000) * (ccx[3]*cspeed - uxloc[k]) + (cclor_y*prodscal - colormod*ecy[3]*cspeed/2.000) * (ccy[3]*cspeed - uyloc[k]);</pre>
	<pre>color_x = 0.000; color_y = (color_field(k]-colorfield(nv4[k])) / delta_x; colormod = sqrt(color_x*color_x + color_y*color_y); prodecal = (color_x * ecx[4] + color_y * ecy[4]) * cspeed; if(color_mod) nff4[k] += neq4[k] * (kforce / (2.000 * colormod)) * ((color_x*prodecal - colormod*ecx[4]*cspeed/2.000) *</pre>
	<pre>(color_y*prodscal - colormod*ecy[4]*cspeed/2.000) * (ecy[4]*cspeed - uyloc[k])); color_x = (colorfield[k]-colorfield[nv3[k]]) / delta_x; color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x; color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x; prodscal = (color_x*color_x + color_y*color_y); prodscal = (color_x * ecx[5] + color_y * ecy[5]) * cspeed;</pre>
	<pre>if(colormod) nff5[k] += neq5[k] * (kforce / (2.000 * colormod)) * (color_x*prodscal - colormod*ecx[5]*cspeed/2.000) * (color_x*prodscal - colormod*ecy[5]*cspeed/2.000) * (color_y*prodscal - colormod*ecy[5]*cspeed/2.000) * (ecy[5]*cspeed - uyloc[k]);</pre>
	<pre>color_x = (colorfield[nv1[k]]-colorfield[k]) / delta_x; color_y = (colorfield[k]-colorfield[nv4[k]]) / delta_x; colormod = sqrt(color_x*color_x + color_y*color_y); prodscal = (color_x * ecx[6] + color_y * ecy[6]) * cspeed; if(color_x * ecx[6] + color_y * colormod) nff6[k] += neq6[k] * (kforce (2.00 * colormod)) * (color_x*prodscal - colormod*ecx[6]*cspeed/2.000) * (color_y*prodscal - colormod*ecx[6]*cspeed/2.000) * (color_y*prodscal - colormod*ecy[6]*cspeed/2.000) * (ecy[6]*cspeed - uxloc[k]) +</pre>
	<pre>color_x = (colorfield[nv1[k]]-colorfield[k]) / delta_x; color_y = (colorfield[nv2[k]]-colorfield[k]) / delta_x; color_mod = sqrt(color_x*color_x + color_y*color_y); prodscal = (color_x * ecx[7] + color_y + color_y); if(color_x * ecx[7] + color_y + color_y * caped; if(color_x*prodscal - colormod*ecx[7]*cspeed/2.000) * (color_x*prodscal - colormod*ecx[7]*cspeed/2.000) * (color_y*prodscal - colormod*ecy[7]*cspeed/2.000) * (color_y*prodscal - colormod*ecy[7]*cspeed/2.000) * (color_y*prodscal - colormod*ecy[7]*cspeed/2.000) *</pre>
	<pre>color_x = (colorfield[k]-colorfield[nv3[k]]) / delta_x; color_y = (colorfield[nv2[k]]-colorfield[k]) / delta_x; color_mod = sqrt(color_x*color_x + color_y*color_y); prodscal = (color_x * ecx[8] + color_y * ecy[8]); if(color_mod) if(color_x*prodscal - colormod*ecx[8]*cspeed/2.000) * (color_x*prodscal - colormod*ecx[8]*cspeed/2.000) * (color_x*prodscal - colormod*ecx[8]*cspeed/2.000) * (color_x*prodscal - colormod*ecy[8]*cspeed/2.000) * (color_x*prodscal - colormod*ecy[8]*cspeed/2.000) *</pre>
**	break;

Printed by sofonea from flumag3.mec.utt.ro	wet9main.c Page 10
Pri	wet9m

Pagin Pagi	Jun 27 1999 13:05 Wet9main.c	Page 9	Jun 27 1999 13:05
	g tg tg		nff7[k] = n - ogradx - ogrady (eoprod[7] nff8[k] = n
11. /* bottom */	pr		- cgradx - cgrady (ecprod[8
Case of the state	*/ break;		break;
Case Case Case Case Case Case Case Case	<pre>case 1: /* bottom */</pre>		77
Call ()	<pre>nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau</pre>	_	- cgradx nff2[k] = n - cgrady
Coa (Coa (Coa (Coa (Coa (Coa (Coa (Coa (<pre>nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau - cgrady * ecq[2] * (nf2[mv2[k]) - nf2[k]) +</pre>		nff3[k] = n - cgradx nff4[k] = n
ca ca			cgrady nff5[k] = n
Coa	<pre>(ecprod[3] - prodscal) * neg3[k]; nf4[k] = nf4[k] - neg4[k])*ctau - cgrady * ecv[4] * (nf4[kv2[k]] - nf4[k]) +</pre>		- cgradx - cgrady nff6[k] = n
coally and the state of the sta	(ecprod[4] - prodscal) * neq4[k]; nff5[k] = nf5[k] - (nf5[k] - nf5[k] - nf5		- cgradx - cgrady
Ca	- ogrady * eck[] * (m.5[k] - m.5[k]]) - cgrady * ecy[5] * (m.5[k]] - m.5[k]) + (ecprod[] - prodes]) * neg5[k];		cgradx
Caa () } }	<pre>nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau</pre>		nff8[k] = n - cgradx - cgrady
(())	<pre>(ecprod[6] - prodscal) * neg6[k]; nff1[k] = nff1[k] - (nf7[k] - neg7[k]) *ctau crack * ecx[7] * (nf7[nu1[k]) - nf7[k])</pre>		break; case 4: /*
()	(F)		nff0[k] = n
} }	(K1)		nff2[k] - cgradx nff2[k] - cgradx
)	roalal		nff3[k] = r - cgradx
) } void first	case 2: /* top */ nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +		nff4[k] = r - cgrady nff5[k] = r
) }	<pre>(ecprod[0] - prodscal) * neg0[k]; nff1[k] = nf1[k] - (nf1[k] - neg1[k]) *ctau</pre>		- cgradx - cgrady nfff[b] = r
} } void first			- cgradx
) } void first	<pre>- cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]) + (ecprod[2] - prodscal) * neq2[k];</pre>		nff7[k] = r - cgradx
	<pre>nff3[k] = nf3[k] (nf3[k]-neq3[k])*ctau</pre>		- cgrady nff8[k] = r
	(ecprod(3) - prodscal) * neg3(k); nff4(k) = nf4(k) - (nf4(k)-neg4(k)*ctau - caradv * ecv(4) * nf4(k) - nf4(nv4(k)) +		- cgradx - cgrady break;
	(ecprod[4] - prodscal) * neq4[k]; nff5[k] = nf5[k] - (nf5[k] - nf5[nv7[k]) - correct * correct		
			void first_centered(c
nff6[k] = nf6[k] - (nf6[k] - nf6[k])	<pre>nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau - cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k]) - cgrady * ecy[6] * (nf6[k] - nf6[nv4[k]]) +</pre>		

Jun 27 1999 13:05 wet9main.c	Page 11
double nff3[], double nff4[], double nff5[], double nff6[], double nff7[], double nff3[], double neq2[], double neq3[], double neq4[], double neq5[], double neq5[], double neq5[], double neq5[], double neq7[], double neq8[], double neq7[], double neq8[],	
int k; double ctau, cgrad, cgrady, cgradky; double ctau, cgrad, cgrady2; double cgradx2, cgradx2; cgradx2; double dimmy_force; double prodscal, color_x, color_y, nn;	
<pre>ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_y; cgrady = cspeed * delta_t / delta_y; cgradxy = cspeed * delta_t / delta_y; cgradxy = cspeed * delta_t / (2.000*delta_x); cgradxy = cspeed * delta_t / (2.000*delta_x); cgradxy2 = cspeed * delta_t / (2.000*delta_y); cgradxy2 = cspeed * delta_t / (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));</pre>	
<pre>for(k=0; k<nnodes_all; k++)<="" td=""><td></td></nnodes_all;></pre>	
[k]) */ - (nf0[k]-neq0[k])*ctau- codscal) * neq0[k])*rtau-	
- cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]) +	
<pre>(ecprod[2] - prodscal) * neq2[k]; nff3[k] = nf3[k] - (nf3[k]-neq3[k]) *ctau - cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]) + (ecprod[13] - prodscal) * neq3[k];</pre>	
<pre>nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]) + (ecprod[4] - prodscal) * neq4[k];</pre>	
<pre>nff5[k] = nf5[k] - (nf5 k]-neq5[k])*ctau</pre>	
<pre>nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau - cgradx2 * ecx[6] * (nf6[nv1]k]] - nf6[nv3[k]]) - cgradx2 * ecy[6] * (nf6[nv2[k]] - nf6[nv4[k]]) +</pre>	
<pre>(ecprod. - prodscal</pre>	
<pre>/* color_x = sforce * (colorfield[nv1[k]]-colorfield[nv3[k]]) / (2.000 * delta_x); color * = sforce * (colorfield[nv2[k]]-colorfield[nv4[k]]) / color * = sforce * (colorfield[nv2[k]]-colorfield[nv4[k]]) / color * = sforce * (colorfield[nv2[k]]-colorfield[nv4[k]]) / color * * * * * * * * * * * * * * * * * * *</pre>	
[2.000 * delta_x); Intf("k=%d colorx=%e colory=%e\n",k,color_x,color_y);	_
nn = nf0[k] + nf1[k] + nf2[k] + nf3[k] + nf4[k] + nf5[k] + nf6[k] + nf6[k] + nf8[k] + nf8[k	+

Printed by sofonea from flumag3.mec.utt.ro

wet9main.c

Jun 27 1999 13:05

<pre>color_x = sforce * nn * (colorfield[nv1[k]]-colorfield[nv3[k]]) / (2.000 * delta_x * mass); color_y = sforce * nn * (colorfield[nv2[k]]-colorfield[nv4[k]]) / (2.000 * delta_x * mass); /*</pre>
lor_x = - sforce * lor_y = - sforce *
- color_a attockal - color_y += neq0[k] * (ecx[0]*cspeed*color_prodscal); += neq1[k] * (ecx[1]*cspeed*color_neq1[k] * (ext[1]*cspeed*color_neq1]*
<pre>nff2[k] += neq2[k] * (ext[2]*cspeed*color_x + ecy[2]*cspeed*color_y -</pre>
+= neq4[k] * (ecx[4]*cspeed*color_x +
<pre>+= neq5[k] * (ecx[5]*cspeed*color_x + prodscal);</pre>
<pre>nff6[k] += neq6[k] * (exx[6]*cspeed*color_x + ecy[6]*cspeed*color_y -</pre>
<pre>prodscal); nff8[k] += neq8[k] * (ecx[8]*cspeed*color_x + prodscal);</pre>
} break;
bottco f [[k]]
case 2: /* top X, top X

Page 14

wet9main.c

Jun 27 1999 13:05

Dimor 27 1999 13:05 Wet9main.c. Page 13:05				
### WETSTRY - (#F2[K] - #F2[K]	age 13			
### ### #### #### #### ###############				
### ### #### #### #### ###############				
		<pre>retau [nv4[k]]) + retau - nf3[nv3[k]]) ectau [nv4[k]]) + retau - nf5[nv3[k]]) fnv4[k]) + retau - nf5[nv3[k]]) fnv4[k]) + retau - nf5[nv3[k]]) fnv4[k]]) + retau</pre>	/* left wall */ k] = nf0[k] - (nf0[k]-neq0[k])*ct k] = nf1[k] - (nf1[k]-neq0[k])*ct gradx * ext[1] * (nf1[k]-neq0[k])*ct grady * ext[1] * (nf1[k]-neq0[k])*ct grady 2 * ext[2] * (nf2[k]-neq0[k])*ct grady 2 * ext[2] * (nf2[k]-neq0[k])*ct grady * ext[2] * (nf2[k]-neq0[k])*ct grady 2 * ext[2] * (nf2[k]-neq0[k])*ct k] = nf5[k] - (nf4[k]-neq0[k])*ct grady 2 * ext[3] * (nf5[nv1[k]] - r grady 2 * ext[3] * (nf5[nv1[k]] - r grady 2 * ext[6] * (nf6[nv1[k]] - r grady 2 * ext[7] * (nf7[nv1[k]] - r grady 2 * ext[8] * (nf8[nv1[k]] - r grady 2	<pre>case 4: /* right wall */</pre>

<pre>nff7(k) = nf7(k) - (nf7(k)-neq7(k))*ctau - cgradx * ecx(7] * (nf7(k) - nf7(nv3(k))) - cgrady2 * ecx(7] * (nf7(nv2(k)) - nf7(nv4(k))); nff8(k) = nf8(k) - (nf8(k)-neg8(k))*ctau - cgradx * ecx(8) * (nf8(k) - nf8(nv3(k))); break; }</pre>
<pre>void second_upwind(double mass, double tau, double cspeed,</pre>
<pre>int k; double ctau, cgrad, cgrady, cgradxy; double grandx2, cgrady2, cgradxy2; double dummy_force; double prodscal;</pre>
<pre>ctau = delta_t / tau;</pre>
<pre>for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>
<pre>prodscal = csforce_x * uxloc[k] + csforce_y * uyloc[k]; switch(boundary_mode[k])</pre>
<pre>case 0: /* bulk */ nff(k = nf0 k - (nf0 k -neq0 k)*ctau + ceprod(0) - prodscal) * neq0 k ; nff(k = nf1 k - (nf1 k -neq1 k)*ctau - cgradx2 * ecx[1] * (3.00*nf1 k - 4.00*nf1 nv3 k) + nf1 nv3 nv3 k]) + (ecprod(11 - prodscal) * neq1 k)*ctau nff2 k = nf2 k - (nf2 k -neq2 k)*ctau</pre>
- cgrady2 * ecy[2] * (3.00%/f2[k]) + nf2[nv4[nv4[k]]]) + (3.00%/f2[k] - 4.00%) * ncm(k])
Vetroit1
)*nff[k] - 4.000*nf5[m33[k]) + nf5[m3][m3[k]]] + ddy2 * evg[*] - 4.000*nf5[m3][k]] + + nf5[m3[m3[k]]]) + + nf5[m4[m4[k]]) + + nf5[m4[m4[k]]) + + nf5[m4[m4[k]]) + + nf5[m4[m4[k]]]) + + nf5[m4[m4[k]]] + nf5[m4[m4[k]]]) + + nf5[m4[m4[k]]] + nf5[m4[m4[k]]]) + + nf5[m4[m4[k]]] + nf5[m4[m4[m4[k]]] + nf5[m4[m4[m4[m4[m4[m4[m4[m4[m4[m4[m4[m4[m4[
<pre>rodscal) * neq5[k]; - (nf6[k]-neq6[k])* + 4.000*nf6[nv1[k]]</pre>
<pre>- cgrady2 * ecy[6] * (3.000*nf6[k] - 4.000*nf6[nv4[k]] + nf6[nv4[k]]]) +</pre>

c.utt.ro	
lumag3.me	
a from f	
sofone	
rinted by	
_	

Jun 27 1999 13:05	wet9main.c	Page 15	Jun 27 1999 13:05
(ecprod(6) - prods nff7[k] = nf7[k] - ((-3.00*nf6[k] + 4] - cgrady2 * ecy[7] + 4] (-3.00*nf7[k] + 4] (ecprod(7) - prods nff8[k] = nf8[k] - 4; (-3.00*nf8[k] - 4; (-3.00*nf8[k] + 4] (-3.00*nf8[k] + 4]	(ecprod[6] - prodscal) * neq6[k]; nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau - cgradk2 * eck[7] * -(3.00%*nf6[k] + 4.000*nf6[nv1[k]]) + -cgrady2 * ecy[7] * -(3.00%*nf7[k] + 4.00*nf7[nv2[k]] - nf7[nv2[nv2[k]]) + (ecprod[7] - prodscal) * neq7[k]; nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau - cgradk2 * eck[8] * (3.000*nf8[k] - (nf8[k]-neq8[k]) * nef7[k]; -cgrady2 * ecy[8] * -cgrady2 * ecy[8] * (ecprod[8] - prodscal) * neq8[k]; hr8[nv2[nv2[k]]]) + (ecprod[8] - prodscal) * neq8[k];	5	ي ر
} } yoid wet9_automaton(void)			IM(SIL), SOURCEL SOURCEL SOURCEL SOURCEL ELLO, E £10, E £1
<pre>int autiter, key_point; autiter = 0;</pre>			source2 source2 ff20, f break;
while (autiter < niter_cycle)	(e.		case 4:
test_distribution_functi test_distribution_functi compute_local_speeds (fil compute_equilibrium_dist switch (key_scheme)	<pre>test_distribution_functions(f10,f11,f12,f13,f14,f15,f16,f17,f18); test_distribution_functions(f20,f21,f22,f23,f24,f25,f26,f27,f28); compute_local_speeds(f10,f11,f12,f13,f14,f15,f16,f17,f18, compute_equilibrium_distributions(); switch(key_scheme)</pre>		break; case 5; 1slb_lines break; case 6; 1slb_upwin
case 0: if (key_init > 2) compute_local_grad compute_upwind_sourc	ddients(); :ces(ElO, fll, fl2, fl3, fl4, fl5, fl6, fl7, fl8, f20, f21, f22, f23, f24, f25, f26, f27, f28);	1. 555,0	case 7: case 8: if (key_ini compute iprop();
upwind_cfl(mass1, t; f10, f11, sf10, f51	<pre>upwind_cfl(mass1, taul, cspeedl, cfl1, fl0, fl1, fl2, fl3, fl4, fl5, fl6, fl7, fl8, fl10, ffl1, ffl2, ffl3, ffl4, ffl5, ffl6, ffl7, ffl8, sfl0, sfl1, sfl2, sfl3, sfl4, sfl5, sfl6,</pre>		break; case 9: lfd(); break;
sfl7, sfl neq10, ne	118, negl1, negl2, negl3, negl4, negl5, negl6,		case 10: iupfd(); break:
ilega., ins sourcel0, sourcel5, mass?, f5	ineq.r, ineq.r, ineq.r, source12, source13, source14, source15, source15, source17, source18);		case 11: iser (); case 12:
upwing_cifidase2, f2, f2, f2, f2, f2, f2, f2, f2, f2, f	upwint_cir(wass.f. faz, f23, f24, f25, f26, f27, f28, f20, f21, f22, f22, f23, f24, f25, f62, ff26, ff27, ff28, sf20, sf21, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28, neq20, neq21, neq22, neq22, neq24, neq25, neq25,	- 1-1/4	ilin(); break; case 13; if (key_in)
source20, source25, break;	source20, source21, source23, source24, source25, source26, source28);		first_upwi
case 1:	1: Sources(f10, f11, f12, f13, f14, f15, f16, f17, f18, [6f10, f21, f21, f22, f23, f24, f25, f26, f27, f28); [6f10, f211, f212, f13, f14, f15, f16, f17, f18, sourcel0, sourcel1, sourcel2, sourcel3, sourcel4,	.,	first_upw
sourcels, sourcel ff10, ff11, ff12, lf(sf20, sf21, sf21, source2, source2,	16, source17, source18, 2, ff13, ff14, ff15, ff16, ff17, ff18); 2, sf23, sf24, sf25, sf26, sf27, sf28, s21, source22, source23, source24,		break; case 14:
source25, source5 ff20, ff21, ff22, break;	source25, source26, source27, source28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28); ak;		if (key_in) compute_ first_cent
case 2: compute_centered_sou	Se 2: compute_centered_sources(f10, f11, f12, f13, f14, f15, f16, f17, f18, f20, f21, f22, f23, f24, f25, f26, f27, f28); centered of (cf11, f10, f11, f12, f13, f14, f15, f16, f17, f18,		

	ff15, ff16, ff17, ff18,	
110, sourcell, sourcell,	source13, source14, source18);	
centered_cfl(cfl2, f20, f21, f22, f23, f24, f25, f26, f2	F26. F27. F28.	
ff20, ff31, ff22, ff23, ff24, ff25, z gurce20, source21, source22, source2 source25, source26, source27, source27,	ff25, ff26, ff27, ff28, source23, source24, source28):	
sources(f10, f11, f12, f13, f20, f21, f22, f23.	6, f17, f18, 6, f27, f28);	
<pre>lw(sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18, source10, source11, source12, source13, source14,</pre>	, sf17, sf18, source14,	
source15, source16, source17, source18,	6617 66101.	
lw(sf20, sf21, sf22, sf25, sf26, sf26, sf26, sf28,	sf27, sf28,	
SOURCEZU, SOURCEZI, SECRI SECRI SECRI SECRI SECRI	sourceza,	
ii20, ii21, ii22, ii23, ii24, ii25, ii26, ii2/, break;	, iid/, iid8);	
case 4:		
isib(); break;		
<pre>case 5: islb linear();</pre>		
ak;		
case 6: islb_upwind();		
break;		
<pre>if (key_init > 2) compute local gradients();</pre>		
iprop();		
Dreak; Case 9:		
lfd();		
Dreak; case 10:		
iupfd();		
Case 11:		
iser();		
Case 12:		
break;		
case 13:		
compute_local_gradients();		
ilrst_upwind(massi, taul; cspeedi, fl0, fl1, fl2, fl3, f14, f15, f16, f	£17, £18,	
ff10, ff11, ff12, ff13, ff14, ff15, neq10, neq11, neq11, neq11, neq12, neq13, neq14, neq12, neq13, neq14, neq13, neq14, neq15, n	red15	
neqi', neqid, echiqui, kiqice, grami first_upwind(mass2, tau2, cspeed2,	gradick, gradicy);	
f20, f21, f22, f23, f24, f25, f26, f27, f28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27,	f26, f27, f28, ff25, ff26, ff27, ff28,	
neg20, neg21, neg22, neg23, neg24, neg24, neg28, ecprod2, -kforce, grad	eq24, neq25, neq26,	
break; case 14;		
<pre>if(key_init > 2) compute_local_gradients();</pre>		
<pre>first_centered(mass1, tau1, cspeed1, f10, f11, f12, f13, f14, f15, f16, f</pre>	f16, f17, f18,	
<pre>ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, neq10, neq11, neq12, neq13, neq14, neq15, neq16,</pre>	ff15, ff16, ff17, ff18, 3q14, neq15, neq16,	
neq17, neq18, ecprod1, kforce);	1	

rinted by sofonea from flumag3.mec.utt.ro	Pace
Printed by s	wet9main c
	27 1000 13:05

Jun 27 1999 13:05 wet9main.c	Page 17	ક
first_centered(mass2, tau2, cspeed2, f25, f26, f27, f28, f20, f21, f22, f23, f24, f25, f26, f27, f28, f20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28 neq22, neq24, neq25, neq25, neq26, ecprod2, -kforce);	6 0	
- 10	ff18,	
	ff28,	
<pre>break;) autiter++; iter++; iter++; test_distribution_functions(ff10, ff11, ff12, ff13, ff14, ff15, ff16, test_distribution_functions(ff20, ff22, ff22, ff23, ff24, ff25, ff26, test_distribution_functions(ff20, ff22, ff23, ff24, ff25, ff26,</pre>		
<pre>if(key_scheme < 7)</pre>	, i (8)	
e e		
<pre>if(key_init > 2) compute_local_gradients(); compute_upwind_gradients(); compute_upwind_gradients(); fill, fill,</pre>		
<pre>upwind_cfl(mass1, tau1, cspeed1, cfl1, ffl0, ffl1, ffl2, ffl3, ffl4, ffl5, ffl6, ffl7, ffl8, fl0, fl1, fl2, fl3, fl4, fl5, fl6, fl7, fl8, sfl0, sfl1, sfl2, sfl3, sfl4, sfl5, sfl6, sfl7, sfl8, neq10, neq11, neq12, neq13, neq14, neq15, neq16,</pre>	- V-04F-FF	
neq17, neq18, source10, source11, source12, source14, source15, source16, source17, source18);		
upwind_cii(mass, fall, file), filed, file,	V (44 - FT)	
break; case 1: lf_sources(ff10, ff11, ff13, ff14, ff15, ff16, ff17, ff18, lf_sources(ff10, ff11, ff28)	•	
10 (11), 12, 13, 14, 115, 116, 117, 118); 11 (120), 8121, 8121, 8123, 8124, 8125, 8126, 8127, 8128, source20, source21, source22, source23, source24, source25, source25, source27, source28, source28,		
break; ter; teo; teo; ter;		
120, 121, 125, 125, 129, 120, 121, 121, 121, 121, 121, 121, 121		

<pre>compute_centered_sources (fil0, ffl1, ffl2, ffl3, ffl4, ffl5, ffl6, ffl7, ffl8,</pre>	
ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18, f10, f11, f12, f13, f14, f15, f16, f17, f18, source10, source11, source13, source14, source15, source15, source16, source18);	
break; case 3: lw_sources(ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,	
source15, source16, source17, source18, f18); f10, f11, f12, f13, f14, f15, f14, f16, f17, f18); f12, s121, s123, s124, s125, s126, s127, s128,	
source20, source21, source22, source23, source24, source25, source25, source26, source27, source	
and the second that the second th	
case 4: [131b();	
Case 5:	
lalb_timear();	
islb_upwind(); break;	
case 7:	
autiter; iter;	
break;	
autiter; iter;	
break;	
compute_equilibrium_distributions();	
break; rase 10:	
compute_local_speeds(ff10,ff11,ff12,ff13,ff14,ff15,f116,f17,ff18,	
compute_equilibrium_distributions();	
ifdprop(); break;	
case 11:	
break	
<pre>case 12: ifdprop();</pre>	
break;	
if (key_init > 2) compute_local_gradients();	
ff20, distri	
first_upwind(mass1, tau1, cspeed1,	

- [
Cioni.

Jun 27 1999 13:05 wet9main.c	Page 19	Jun 27 1999 13:05
ffl0, ffl1, ffl2, ffl3, ffl4, ffl5, ffl6, ffl7, ffl8, fl0, fl1, fl2, fl3, fl4, fl5, fl6, fi7, fl8, neq10, neq11, neq12, neq13, neq14, neq15, neq16, neq17, neq18, ecprod1, kforce, gradn2x, gradn2y);		delta_t = a_dell force_x = a_for force_y = a_for gforce = a_gfor
first_upwind(mass2, tau2, cspeed2, ff23, ff26, ff26, ff27, ff28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, f20, f21, f22, f23, f24, f25, f26, f27, f28, neq21, neq22, neq24, neq25, neq26, neq26, neq26, neq26, neq26, neq26, neq26, neq26, neq26, neq28, neq28, neq28, neq26, neq28,		massi = a_massi mass2 = a_mass2 cspeed1 = a_csp cspeed2 = a_csp ++++++++++++++++++++++++++++++++++++
break; case 14: if (key_init > 2) compute local gradients();		tau2 =
compute_local_speeds (fil0, ff11, ff12, ff14, ff15, ff16, ff15, ff18, ff18, compute_equilibrium_distributions();		nzero2left = a_nzero1right = a_nzero2right = a
first_centered(mass1, tau1, cspeed1, ff15, ff16, ff17, ff18, ff12, ff13, ff14, ff15, ff6, f11, f18, f10, f11, f12, f13, f14, f15, f16, f11, f18, neq10, neq10, neq11, neq12, neq13, neq14, neq15, neq16,		<pre>uxwall_bot = a_ uywall_bot = a_ uxwall_top = a_ uywall_top = a_</pre>
first_centered(mass2, tau2, cspeed2, ff26, ff26, ff27, ff28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, ff20,	• 100	uxwall_left = a uywall_left = a uxwall_right = uywall_right =
negzu, negzz, negzz, negzz, negzz, negzz, negz7, negz8, ecprodz, -kforce); break;		cspeed12 = cspe
case 15: second_upwind(mass1, tau1, cspeed1, second_upwind(mass1, tau1, ff12, ff13, ff14, ff15, ff16, ff17, ff18, f10, f11, f12, f13, f14, f15, f16, f17, f18, neq10, neq11, neq12, neq14, neq15, neq15,		ctaul = delta_t ctau2 = delta_t csforce_x = del csforce_y = del
second_upwind(mass2, tau2, cspeed2, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, f20, f21, f22, f23, f24, f25, f26, f27, f28, f20, f21, f22, f23, f24, f25, f26, f27, f28,		kforce = delta_ kbound1 = delta_ kbound2 = delta
neq27, neq22, neq22, neq23, neq23, neq23, neq20, neq26, neq28, ecprod2);		cfl1 = cspeedl cfl2 = cspeed2
autiter++; iter++;		oneminus1 = 1.0 oneplus1 = 1.00
		oneminus2 = 1.0
<pre>void main(void) { int isim, icycle;</pre>		uplminus2 = cfl uplminus1 = cfl up1 = (2.000 -
<pre>FilE *frez; wet9 input(); for(isim=0; isim<nsim; isim++)<="" pre=""></nsim;></pre>		up2minus2 = cfl up2minus1 = cfl up2 = (2.000 -
<pre>nnodes_x = a_nnodes_x[isim]; nnodes_y = a_nnodes_y[isim]; nnodes_all = a_nnodes_all[isim];</pre>		in1 = (1.000 + in1plus1 = - of in1plus2 = cfl1
<pre>lambda = a_lambda[lsim]; key_nint = a_key_int[lsim]; key_boundary = a_key_boundary[lsim]; key_scheme = a_key_scheme[lsim];</pre>	-	in2 = (1.000 + in2plus1 = - of in2plus2 = cfl2
<pre>key_interface = a_key_interface[isim]; ncycles = a_ncycles[isim]; niter_cycle = a_niter_cycle[isim]; niter_init = a_niter_init[isim];</pre>		tlminus = cfll tlcenter = (1.(tlplus = (cfll
<pre>length_x = a_length_x[isim]; length_y = a_length_y[isim]; delta_x = a_delta_x[isim]; delta_x = a_delta_x[isim];</pre>		t2minus = cfl2 t2center = (1.0 t2plus = (cfl2

<pre>delta_t = a_delta_t[isim]; force_x = a_force_k(isim); force_y = a_force_y(isim); gforce = a_gforce[isim]; massl = a_massl[isim]; mass2 = a_mass2[isim]; cspeedi = a_cspeedi[isim];</pre>	
48877"	
nzerožleft = a_nzerožleft [isim]; nzerozright = a_nzerozright[isim]; nzerozright = a_nzerozright[isim]; uxwall_bot = a_uxwall_bot [isim]; uxwall_bot = a_uxwall_bot [isim]; uxwall_top = a_uxwall_top[isim]; uxwall_top = a_uxwall_top[isim]; uxwall_left = a_uxwall_top[isim];	
eft ight ight	
= cspeed2 * delta_t / tau delta_t / tau x = delta_t ' y = delta_t '	
<pre>kforce = delta_t * gforce / (kboltz * temp); kbound1 = delta_t; kbound2 = delta_t;</pre>	
<pre>cfil = cspeed1 * delta_t / delta_x; cfi2 = cspeed2 * delta_t / delta_x;</pre>	
oneminus1 = $1.000 - cf11;$ oneplus1 = $1.000 + cf11;$	
oneminus2 = 1.000 - cfl2; oneplus2 = 1.000 + cfl2;	
uplminus2 = cfil * (cfil - 1.000) / 2.000; uplminus1 = cfil * (2.000 - cfil); upl = (2.000 - cfil) * (1.000 - cfil) / 2.000;	
up2minus2 = cf12 * (cf12 - 1.000) / 2.000; up2minus1 = cf12 * (2.000 - cf12); up2 = (2.000 - cf12) * (1.000 - cf12) / 2.000;	
<pre>inl = (1.000 + cfl1) * (2.000 + cfl1) / 2.000; inlplus1 = - cfl1 * (2.000 + cfl1); inlplus2 = cfl1 * (2.000 + cfl1) / 2.000;</pre>	
in2 = (1.000 + cfl2) * (2.000 + cfl2) / 2.000; in2plus1 = - cfl2 * (2.000 + cfl2); in2plus2 = cfl2 * (2.000 + cfl2) / 2.000;	1 2 2
tlminus = cfl1 * (1.000 + cfl1) / 2.000; tlcenter = (1.000 - cfl1) * (1.000 + cfl1); tlplus = (cfl1 - 1.000) * cfl1 / 2.000;	
t2minus = cfl2 * (1.000 + cfl2) / 2.000; t2center = (1.000 - cfl2) * (1.000 + cfl2); t2plus = (cfl2 - 1.000) * cfl2 / 2.000;	

.utt.ro	
3.тес	
lumag(
ea from f	
y sofonea	
ed by so	
Printed	
۵	

wet9main.c

dun 27 1999 13:05	/* xv_new(£10. */ te8t_ntet.(11 (key_ini wet9_cha if (key_ini wet9_cou	/* quiver();	free_nsim();															
Page 21																			
Jun 27 1999 13:05 wet9main.c	tin1 = (1.000 + cf11) * (2.000 + cf11) / 2.000; tin1plus1 = - cf11 * (2.000 + cf11); tin1plus2 = cf11 * (1.000 + cf11) / 2.000;	tin2 = (1.000 + cf12) * (2.000 + cf12) / 2.000; tin2plus1 = - cf12 * (2.000 + cf12); tin2plus2 = cf12 * (1.000 + cf12) / 2.000;	<pre>tout1 = (1.000 - cfil) * (2.000 - cfll) / 2.000; toutlminus1 = cfll * (2.000 - cfll); toutlminus2 = cfll * (cfll - 1.000) / 2.000;</pre>	<pre>tout2 = (1.000 - cf12) * (2.000 - cf12) / 2.000; tout2minus1 = cf12 * (2.000 - cf12); tout2minus2 = cf12 * (cf12 - 1.000) / 2.000;</pre>	tlfminusl = (1.000 + cfll) / 2.000; tlfplusl = (1.000 - cfll) / 2.000;	tlfminus2 = (1.000 + cfl2) / 2.000; tlfplus2 = (1.000 - cfl2) / 2.000;	<pre>isminus1 = cfl1 * (1.000 + cfl1) / 2.000; iscenter1 = (1.000 - cfl1) * (1.000 + cfl1); isplus1 = cfl1 * (cfl1 - 1.000) / 2.000;</pre>	<pre>isminus2 = cfl2 * (1.000 + cfl2) / 2.000; iscenter2 = (1.000 - cfl2) * (1.000 + cfl2); isplus2 = cfl2 * (cfl2 - 1.000) / 2.000;</pre>	<pre>build_names(isim); init_lattice_functions(); init_ecx_nine(); init_arrays_nine_square(); init_arrays_nine_square_aux(); getavec_square();</pre>	iter = 0;	<pre>xx_new(fl0,fl1,fl2,fl3,fl4,fl5,fl6,fl7,fl8,xv_name,1.0001); xv(fl0,fl1,fl2,fl3,fl4,fl5,fl6,fl7,fl8,"xv");</pre>	if (X	xv (f10, f11, f12, f13, f14, f15, f16, f17, f18, "XV1"); xv (f20, f21, f22, f23, f24, f25, f26, f27, f28, "XV2");	<pre>if(key_init == 3) wet9_drop_profile(); if(key_init == 9) wet9_drop_profile(); test_ntot();</pre>	<pre>if(key_init<2) wet9_chant profile(); if(key_init == 2) wet9_couple_profile();</pre>	<pre>for(icycle=0; icycle</pre> icycle++)	<pre></pre>	xv(£10,£11,£12,£13,£14,£15,£16,£17,£18,"xV1"); xv(£20,£21,£22,£23,£24,£25,£26,£27,£28,"xV2");	<pre>if(key_init == 3) (key_init == 9)) wet9_drop_profile();</pre>

				<u> </u>			 	
,1.0001);								
/* *<_new(f10,f11,f12,f13,f14,f15,f16,f17,f18,xv_name,1.0001);								
, f13, f14, f15, f16	file(); ile();	<i>:</i>						
new (f10, f11, f12,	<pre>test_ntot(); if(key_init<2) wet9_channel_profile(); if(key_init == 2) wet9_couple_profile(); }</pre>	/* quiver(); */ free_lattice_functions();						
*/x			free_nsim();					

Appendix B dif9 code

dif9head.h

May 3 1999 16:15 dif9head.h	Page 1	May 3 1999 16:15
*******************		double *ff10, *ff1:
* dif9head.h definition of global variables *		double *ff20, *ff2
** ** ** ** ** ** ** **		double *sf10, *sf1
#ifdef Main_HEADER		double *sf20, *sf2
const double kboltz=1.381e-23, amu=1.661e-27, temp=300.00;	-	double *neq10, *neq
/* kboltz = $J/$ (molecules*K), amu = (kg), temp = (K); */		double *neq20, *neq
<pre>const double w0 = ((double) 4) / ((double) 9), w1 = ((double) 1) / ((double) 9), w2 = ((double) 1) / ((double) 36);</pre>		double *uxloc, *uy int *nv1, *nv2, *n
<pre>const double three = ((double) 3), three_over_two = ((double) 3) / ((double) 2), nine over two = ((double) 9) / ((double) 2);</pre>		double *nloc1, *nl
const double uxwall = 0.00000, uywall = 0.00000;		extern const doubl
int nnodes_x, nnodes_y, nnodes_all, lambda;	-	extern const doubl
int *a_nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda;	-	extern const doubl
int key_init, key_boundary, key_g, key_force, key_point, key_scheme;		extern const doubl
int *a_key_init, *a_key_boundary, *a_key_g, *a_key_force, *a_key_scheme;		extern int nnodes
int nsim, ncycles, niter_cycle, niter_init, iter, niter;		extern int *a_nnod
int *a_ncycles, *a_niter_cycle, *a_niter_init;		extern int key_ini
<pre>double length_x, length_y, delta_x, delta_y, delta_t,</pre>		extern int *a_key_
force_X, force_Y;		extern int hsim, n
double "a_tengui_k, "a_tengui_y, a_terca_k, a_terca_y, a_terca_t, a_force_y;		extern double leng
double mass1, mass2, tau1, tau2, dcoef, alpha, alpha1;		fuer aronom manya
<pre>double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2,</pre>		forc
double nzeroileft, nzero2left, nzerolright, nzero2right;		extern double *a_l
double *a_nzerolleft, *a_nzero2left, *a_nzerolright, *a_nzero2right;		extern double mass
<pre>char input_name[] = "dif9.input", output_name[] = "dif9.output";</pre>		extern double *a_m
char id_name[128], rez_name[128], xv_name[128];		extern double nzer
double ecx[9], ecy[9], ecxx[9], ecxy[9], ecyx[9], ecyy[9], edxy;		r e* elfinor metve
double ecx1[9], ecy1[9], ecxx1[9], ecxy1[9], ecyx1[9], ecyy1[9];		owtern char tannit
double ecx2[9], ecy2[9], ecxx2[9], ecxy2[9], ecyx2[9];		extern char id nam
double ec1[9], ec2[9];		extern double ecx[
,		extern double ecx1
*£12, *£13, *£14, *£15, *£16, *£17,		extern double ecx2
double *f20, *f21, *f22, *f23, *f24, *f25, *f27, *f28;		extern double ecl[

double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16, *ff17, *ff18;
double *ff20, *ff21, *ff22, *ff23, *ff24, *ff25, *ff26, *ff27, *ff28;
double *sf10, *sf11, *sf12, *sf13, *sf14, *sf15, *sf16, *sf17, *sf18;
double *sf20, *sf21, *sf22, *sf23, *sf24, *sf25, *sf26, *sf27, *sf28;
double *neq10,*neq11,*neq12,*neq13,*neq14,*neq15,*neq16,*neq17,*neq18;
double *neq20,*neq21,*neq22,*neq23,*neq24,*neq25,*neq26,*neq27,*neq28;
double *uxloc, *uyloc, *uxloc1, *uyloc1, *uxloc2, *uyloc2;
int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;
double *nloc1, *nloc2, *rholoc1, *rholoc2, *rholoc;
#e1se
extern const double kboltz, amu, temp;
extern const double w0, w1, w2;
extern const double three, three_over_two, nine_over_two;
extern const double uxwall, uywall;
extern int nnodes_x, nnodes_y, nnodes_all, lambda;
extern int *a_nnodes_x, *a_nnodes_y, *a_nnodes_all, *a_lambda;
extern int key_init, key_boundary, key_g, key_force, key_point, key_scheme;
extern int *a_key_init, *a_key_boundary, *a_key_g, *a_key_force, *a_key_scheme;
extern int nsim, ncycles, niter_cycle, niter_init, iter, niter;
extern int *a_ncycles, *a_niter_cycle, *a_niter_init;
extern double length_x, length_y, delta_x, delta_y, delta_t, cspeed1, cspeed2, cspeed22, cspeed22, cspeed22, force_d1s2, cspeed2s2, force_y;
extern double *a_length_x, *a_length_y, *a_delta_x, *a_delta_y, *a_delta_t, *a_force_x, *a_force_y;
extern double mass1, mass2, tau1, tau2, dcoef, alpha, alpha1;
extern double *a_mass1, *a_mass2, *a_cspeed1, *a_cspeed2, *a_tau1, *a_tau2, *a_dcoef;
extern double nzerolleft, nzerolleft, nzerolright, nzerolright;
extern double *a_nzerolleft, *a_nzero2left, *a_nzero1right, *a_nzero2right;
extern char input_name[], output_name[];
extern char id_name[], re2_name[], xv_name[];
extern double ecx[], ecy[], ecxx[], ecxy[], ecyx[], ecyy[], edxy;
extern double ecx1[], ecy1[], ecxx1[], ecyx1[], ecyx1[], ecyy1[];
extern double ecx2[], ecy2[], ecxx2[], ecxy2[], ecyx2[], ecyy2[];
extern double ecl[], ec2[];

extern int 'boundary_mode; extern double *[10, *[11, *[12, *[13, *[14, *[15], *[16, *[17, *[18]; extern double *[20, *[21, *[22, *[23], *[24, *[25], *[26], *[27], *[28]; extern double *[20, *[21, *[22, *[23], *[24, *[25], *[26], *[27], *[28]; extern double *[20, *[21], *[22, *[23], *[24, *[25], *[26], *[27], *[28]; extern double *[20, *[21], *[22, *[23], *[24], *[25], *[26], *[28]; extern double *[20, *[21], *[22, *[23], *[24], *[25], *[26], *[28]; extern double *[20, *[21], *[22, *[23], *[24], *[24], *[24], *[28]; extern double *[20, *[21], *[22], *[24], *[24], *[24], *[24], *[28]; extern double *[20, *[21], *[21], *[21], *[21], *[22], *[24], *[28]; extern double *[20, *[21],	May 3 1999 16:15 dif9head.h	Page 3
extern double *fi0, *fi1, *fi3, *fi3, *fi6, *fi6, *fi7, *fi8; extern double *fi0, *fi1, *fi2, *fi3, *fi3, *fi16, *fi17, *fi18; extern double *fi10, *fi11, *fi12, *fi23, *fi24, *fi16, *fi16, *fi17, *fi18; extern double *fi10, *fi11, *fi22, *fi23, *fi24, *fi26, *fi16, *fi17, *fi28; extern double *fi10, *fi11, *fi12, *fi13, *fi14, *fi15, *fi16, *fi17, *fi18; extern double *fi10, *fi11, *fi12, *fi13, *fi14, *fi15, *fi16, *fi18; extern double *fineq10, *neq11, *neq12, *neq13, *neq15, *neq16, *neq17, *neq18; extern double *nunco, *tyloc, *uxloc1, *tyloc1, *uxloc2, *rug20, *neq21, *neq22, *neq21, *neq22, *neq22, *neq21, *neq15, *fi16c2;; extern double *nunco, *tyloc, *uxloc1, *tyloc1, *uxloc2;; extern double *nloc1, *nloc2, *tholoc2, *tholoc2; #endif	extern int *boundary_mode;	
extern double *f20, *f21, *f23, *f23, *f24, *f25, *f26, *f21, *f218, *f216, *f217, *f218, *f216, *f217, *f218, *f217, *f218, *f217, *f218, *f217, *f218, *f217, *f218, *f217, *f218, *f217, *f217, *f218, *f217, *f218, *f217, *f218, *f217, *f218, *f2		
extern double *ff10, *ff11, *ff12, *ff13, *ff14, *ff15, *ff16, *ff27, *ff28, *f	£28;	
extern double *ff20, *ff21, *ff23, *ff24, *ff25, *ff16, *f511, *f518; extern double *f10, *f111, *f112, *f113, *f114, *f115, *f116, *f117, *f118; extern double *f20, *f221, *f222, *f223, *f224, *f255, *f256, *f267, *f268; extern double *neq10,*neq11,*neq12,*neq24,*neq25,*neq26,*neq27,*neq28; extern double *valoc, *va		
<pre>extern double *sf10, *sf11, *sf12, *sf13, *sf14, *sf16, *sf17, *sf18; extern double *sf20, *sf21, *sf22, *sf24, *sf25, *sf26, *sf27, *sf28; extern double *neg10,*neg11,*neg12,*neg13,*neg16,*neg16,*neg17,*neg18; extern double *uxloc, *uyloc, *uxloc1, *uyloc1, *uyloc2, *uyloc2;; extern double *uxloc, *uyloc, *uxloc1, *uyloc1, *uyloc2; *uyloc2;; extern it *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8; extern double *nloc1, *nloc2, *xholoc1, *tholoc2; #endif</pre>		
extern double *sf20, *sf21, *sf22, *sf23, *sf26, *sf26, *sf26, *sf27, *sf28; extern double *neq10, *neq10, *neq10, *neq10, *neq10, *neq20, *neq21, *neq22, *neq23, *neq24, *neq25, *neq26, *neq27, *neq28; extern double *neq20, *neq21, *neq23, *neq24, *neq25, *neq26, *neq21, *neq28; extern int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv1, *nv8; *nv1, *nv2, *nv2, *nv2, *nv5, *nv6, *nv6, *nv1, *nv8; *nv1, *nv2, *nv1, *nv2, *nv6, *nv6, *nv6; *nv1, *nv8; *nv1, *nv1, *nv1, *nv2, *nv1, *nv2, *nv1, *nv2, *nv1, *nv2, *nv1, *nv2, *nv2, *nv1, *nv2, *n		
<pre>extern double *neq10,*neq11,*neq12,*neq13,*neq14,*neq15,*neq16,*neq16,*neq26,*neq</pre>		
extern double *neq20,*neq21,*neq22,*neq23,*neq26,*neq26,*neq28; extern double *uxloc, *uyloc, *uyloc], *uyloc2; ; extern int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6; extern double *nloc1, *nloc2, *rholoc1, *rholoc2, *rholoc; #endif	extern double *neq10,*neq11,*neq12,*neq13,*neq14,*neq15,*neq16,*neq17,*neq18;	
<pre>rn double *uxloc, *uyloc, *uyloc; *uyloc; *uyloc; rn int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8; rn double *nloc; *nloc; *rholoc; *rholoc;</pre>	extern double *neq20, *neq21, *neq22, *neq23, *neq24, *neq25, *neq26, *neq27, *neq28;	
extern double *nloc1, *nloc2, *rholoc2, *rholoc; *endif	extern int *nv1, *nv2, *nv3, *nv4, *nv5, *nv6, *nv7, *nv8;	
. ·	extern double *nloc1, *nloc2, *rholoc1, *rholoc2, *rholoc;	
	#endif	
·		

dif9inout.c

May 5 1999 09:19

May 5 1999 09:19 dif9inout.c	Page 1
/*************************************	
<pre>#include <stdio.h> #include <stdib.h> #include <string.h> #include <arting.h> #include <math.h></math.h></arting.h></string.h></stdib.h></stdio.h></pre>	
#include "dif9head.h" word allocate neimfundd	
defined and a control of the property of the p	•
free(a_mass1);	

<pre>free(a_mass2); free(a_cspeed1); free(a_cspeed2); free(a_tau2); free(a_tau2); free(a_necolleft); free(a_necolleft); free(a_necolleft); free(a_necolleft); free(a_necolleft); free(a_necolleft); free(a_necolleft);</pre>
<pre>void dif9_input (void) { int i, j; char dummy[128]; /* double kboltz=1.381e-23, amu=1.661e-27, temp=300.00; kboltz = J/(molecules*K), amu = (kg), temp - (K); */ FILE *fin, *frez; frez=fopen (output_name, "w"); frez=fopen (output_name, "w");</pre>
<pre>if((fin = fopen(input_name, "r")) == NULL) fprintf(frez, "program <dif9> stopped - input file <%s> does not exist !\n", folose(frez); exit(1); fscanf(fin, "%s %d\n", dummy, ≁); if(nsim < 1) exit(1); allocate_nsim(); allocate_nsim();</dif9></pre>
for(i=0; i <nsim; *\$<="" f="" i++)="" scanf(fin,="" td=""></nsim;>

dif9inout.c

May 5 1999 09:19

	May 5 1999 09:19 dif9inout.c Page 3
= (4.00 * kboltz * temp) / (mass1 * amu); = sqrt(cspeed1); = sqrt(cspeed1); = sqrt(cspeed2); = sqrt(cspeed2);	į ÷
<pre>((double) lambda)*delta_t*cspeed1 > delta_x) ((double) lambda)*delta_t*cspeed2 > delta_x) ((double) lambda)*delta_t*cspeed2 > delta_x) (((double) lambda)*delta_t*cspeed2 > delta_x) (((double) lambda)*delta_t*cspeed2 > delta_y) printf("*cdif9> stopped - small delta_x or delta_y !\n"); fprintf(fcrez,"cdif9> stopped - small delta_x or delta_y !\n"); exit(1); call (delta_t > taul) (delta_t > tau2)) printf("*cdif9> stopped - delta_t > tau or tau2 !\n"); fprintf(fcrez,"cdif9> stopped - delta_t > tau or tau2 !\n"); exit(1); exit(1);</pre>	= (4.00 * kboltz * temp) / (massl * = (4.00 * kboltz * temp) / (mass2 * = sqrt(cspeedl); = sqrt(cspeed2);
<pre>printf("<dif9> stopped - small delta_x or delta_y !\n"); fcprintf(frez,"<dif9> stopped - small delta_x or delta_y !\n"); fcprintf(frez,"<diff> stopped - small delta_x or delta_y !\n"); exit(!); (delta_t > taul) (delta_t > tau2)) printf("<dif9> stopped - delta_t > tau 1 or tau2 !\n"); fclose(frez); exit(!);</dif9></diff></dif9></dif9></pre>	(((double) lambda)*delta_t*cspeedl > (((double) lambda)*delta_t*cspeedl > (((double) lambda)*delta_t*cspeed2 > (((double) lambda)*delta_t*cspeed2 >
<pre>(delta_t > tau1) (delta_t > tau2)) printf("<dif9> stopped - delta_t > tau 1 or tau2 !\n"); fprintf(frez," <dif9> stopped - delta_t > tau 1 or tau2 folose(frez); exit(1);</dif9></dif9></pre>	- small delta_x or delta_y !\n"); :opped - small delta_x or delta_y !\n")
	<pre>(delta_t > tau1) (delta_t > tau2)) printf("<dif9> stopped - delta_t > tau 1 or tau2 !\n"); fprintf(frez,"<dif9> stopped - delta_t > tau 1 or tau2 exit(1);</dif9></dif9></pre>
<pre>nnodes_all = nnodes_x * nnodes_y; length_x = delta_x * ((double) (nnodes_x-1)); length_y = delta_y * ((double) (nnodes_y-1));</pre>	<pre>nnodes_all = nnodes_x * nnodes_y; length_x = delta_x * ((double) (nnodes_x-1)); length_y = delta_y * ((double) (nnodes_y-1));</pre>
<pre>fprintf(frez,"%d %d %lf %lf %lf\n",nnodes_x,nnodes_y, nnodes_all,length_x,length_y,delta_t); fprintf(frez,"kboltz= %e %lf %lf %lf %e\n",kboltz,temp,mass1,mass2,amu);</pre>	<pre>fprintf(frez,"%d %d %lf %lf %lf\n",nnodes_x,nnodes_y, nnodes_all,length_x,length_y,delta_t); fprintf(frez,"kboltz= %e %lf %lf %lf %e\n",kboltz,temp,mass1,mass2,amu);</pre>
<pre>a_nnodes_x[i] = nnodes_x; a_nnodes_y[i] = nnodes_y; a_nnodes_y[i] = nnodes_y; a_lambda[i] = lambda[i] = key_init; a_key_init[i] = key_init; a_key_loundary[i] = key_init; a_key_corec[i] = key_init; a_nnier_core[i] = key_core; a_nier_core[i] = key_core; a_nier_core[i] = nier_cycle; a_nier_init[i] = nier_cycle; a_nier_init[i] = nier_init; a_length_x[i] = length_x; a_length_x[i] = length_x; a_delta_t[i] = delta_x; a_delta_t[i] = delta_x; a_delta_t[i] = delta_x; a_delta_t[i] = delta_x; a_delta_t[i] = nength_x; a_nnerollet[i] = nerollet[i] a_nnerollet[i] = nerollight; a_nnerollet[i] = nerollight; fclose(fin); fclose(fin);</pre>	<pre>a_nnodes_x[i] = nnodes_x; a_nnodes_x[i] = nnodes_y; a_nnodes_gall[i] = nnodes_gl]; a_key_lni[i] = lambda; a_key_init[i] = key_init; a_key_lnit[i] = key_init; a_key_force[i] = key_scheme; a_ney_force[i] = ney_les; a_niter_ord[i] = ney_les; a_niter_ord[i] = nney_les; a_niter_init[i] = niter_init; a_length_x[i] = length_x; a_length_x[i] = length_x; a_length_x[i] = length_x; a_delt_a_x[i] = delt_a_x; a_delt_a_x[i] = length_x; a_corce_x[i] = force_x; a_corce_x[i] = force_x; a_corce_x[i] = force_x; a_corce_x[i] = corce_x; a_nass[i] = taus; a_nass[i] = taus; a_nasrolleft[i] = nasrolleft; a_nerolleft[i] = nasrolleft[i] = nasrolleft</pre>

void dif9_output (void)	
<pre>int i; FILE *fout; fout = fopen(output_name,"w"); fpxintf(fout,"nsim= *d\n",nsim); for(1=0; i<nsim; i++)<="" pre=""></nsim;></pre>	
<pre>fprintf(fout,"nnodes_x= %d nnodes_y= %d\n",a_nnodes_x[i],a_nnodes_y[i]); fprintf(fout,"delta_x= %if delta_y= %if delta_t= %if\n",</pre>	
fclose(fout); }	
Int imass, inass2, itaul, itau2, icspeed, icorea, idcoce, idelea_x, idelea_x	94
	•

Jun 20 1999 18:52	diryinit.c	rage 1
*********	**************************************	
* dif9init.c arrays initialisation	alisation *	
************************	***************************************	
<pre>#include <stdio.h> #include <stdlib.h> #include <math.h> #include <string.h></string.h></math.h></stdlib.h></stdio.h></pre>		
#include "dif9head.h"		
void print_tavec(int kk, int nv[])		
<pre>int i, j; FILE *fout; fout = fopen(rez_name, "aw"); forint(fout, "\nnv*ld\n", kk); for(j=0;j<=nnodes_y;j++)</pre>		
{ for (1=0; 1<=nnodes_x; 1++)		
<pre>fprintf(fout, "%3d ", nv[(j-1)*nnodes_x+i]);</pre>	v[(j-1)*nnodes_x+i]);	
<pre>fprintf(fout, "\n");</pre>		
fclose(fout);		
void getavec_square(void)		
$\inf_{\mathbf{k}} \mathbf{i}_{n} \mathbf{j}_{r} \mathbf{k}_{r}$		
_		
for(i=0; i <nnodes_x; i++)<="" td=""><td></td><td></td></nnodes_x;>		
<pre>inv1[k] = k+1; if(i == (nnodes_x-1))</pre>	11)	
nv3(k) += nnodes_x; nv4(k) = k-nnodes_x; if(nv4(k) < 0)		
nv4[k]+= nnodes_all; nv5[k] = nv2[k] + 1; if(i == (nnodes x-1))		
nv5[k] -= nnodes_x; nv6[k] = nv2[k] - 1;		
nv6[k] += nnodes_x; nv7[k] = nv4[k] - 1; if (i == 0)		
nv7 [k] += nnodes_x; nv8 [k] = nv4 [k] + 1; if (i == (nnodes_x)) nv8 [k] -= nnodes_x;		
/* printf("k=%d %d %d k,nv1[k],nv2[k	/* print["k=%d %d %d %d %d %d %d %d\n", print["k=%d %d %d %d %d %d\l", nv5[k], nv8[k]); k,nv1[k],nv2[k],nv3[k],nv4[k],nv6[k],nv6[k],nv7[k],nv8[k]);	,,
*/ K++;		

dif9init.c

form(i),ecx(i),ecx(i),ecy(i),h\$3d %lf %lf %lf %lf\n",
i,ecx(i),ecx(i),ecy(i),ecy(i));
fprint(frez, "1,ecx(i),ecy1(i),n\$3d %lf %lf\n",
i,ecx(i),ecy1(i));

fprintf(frez,"i,ecx[i],ecy[i]\n%3d %lf %lf\n",
 i,ecx[i],ecy[i]);
fprintf(frez,

frez=fopen(rez_name, "a");

for(i=0; i<9; i++)

dif9init.c

	Jun 20 1999 18:52	dif9init.c	Page 3
<pre>innodes_y; j++) i < nnodes_x; j++) =-nnodes_x; j++) =-nnodes_x; j++) =-nnodes_x; j++) ((double) = nzero* (1. +rand_coef * (double) = nzero* (1.0 +rand_coef * (double) = (double) = nzero* (1.0 +rand_coef * (double) = (double) = nzero* (1.0 +rand_coef * (double) = (doubl</pre>	<pre>fprintf(frez, "1, ecx 1, ecx2[1], e }</pre>	Z[i],ecyz[i]\n%3d %lf %lf\n", cy2[i]);	
coef * (double) RAND_MAX) - 0 1. +rand_coef * ble) RAND_MAX) - 0.5)); Lcoef * (double) RAND_MAX) - 0 +rand_coef * (double) RAND_MAX) - 0 - diffusion */ - diffusion */ - trand_coef * (double) RAND_MAX) - 0 - diffusion */ - diffusion */ (double) RAND_MAX) - 0 (double) RAND_MAX) - 0 (in +rand_coef * ble) RAND_MAX) - 0.5);	switch (key_init)		
* (1.0 +rand_coef * (double) RAND_MAX) - 0 * (1.0 +rand_coef * (double) RAND_MAX) - 0.5)); * (1. +rand_coef * - 0.5)); +rand(coef * (double) RAND_MAX) - 0.5)); * (1. +rand_coef * - 0.5)); * (1. +rand_coef * - 0.5)); * (1. +rand_coef * - 0.5)); * (1.0 +rand_coef * - 0.5)); * (1.0 +rand_coef * - 0.5)); * (double) RAND_MAX) - 0.5); * (double) RAND_MAX) - 0.5); * +rand_coef * - 0.5);	case 0: /* horizontal k=-1; for (j=0; j< nnode	<pre>diffusion couple */ s_y; j++)</pre>	
rand_coef * * (1.0 + rand_coef *	for(i=0; i < n	nodes_x; i++)	
* (1.0 +rand_coef * (double) RAND_MAX) - 0 * (1.0 +rand_coef * 0.5)); * (1. +rand_coef * 0.5)); +rand_coef * (double) RAND_MAX) - 0 rand() / (double) RAND_MAX) - 0 self - diffusion */ self - diffusion */ * (1. +rand_coef * 0.5)); * (double) RAND_MAX) - 0 * (double) RAND_MAX) - 0.5)); * * (1. +rand_coef * 0.5)); * * * * * * * * * * * * * * * * * * *	k++; if(1<=nnode	6_x/2)	,
* (1.0 +rand_coef * (double) RAND_MAX) - 0.5)); * (1. +rand_coef * rand_coef * rand() / (double) RAND_MAX) - 0.5)); +rand_coef * rand() / (double) RAND_MAX) - 0.5 self - diffusion */ rand() / (double) RAND_MAX) - 0 * (1.0 +rand_coef * rand_coef * (double) RAND_MAX) - 0 * (1.0 +rand_coef * rand_coef * (double) RAND_MAX) - 0 * (1.0 +rand_coef * rand_coef * (double) RAND_MAX) - 0.5)); * (1.1 +rand_coef * rand_coef * rand() / (double) RAND_MAX) - 0.5)); * * (1.1 +rand_coef * rand_coef * rand() / (double) RAND_MAX) - 0.5)); * * * * * * * * * * * * * * * * * * *	f10[k]	nzero * (1. +rand_coef * ((double) rand() / (double) RAND_MAX) - 0.0e+0;	
* (1. +rand_coef * ' (double) RAND_MAX) - 0.5)); +rand_coef * rand() / (double) RAND_MAX) - 0. ' (1. +rand_coef * rand() / (double) RAND_MAX) - 0. self - diffusion */ rand() / (double) RAND_MAX) - 0. ' (double) RAND_MAX) - 0.5); ' (double) RAND_MAX) - 0.5); +rand_coef * ' (double) RAND_MAX) - 0.5); +rand_coef * rand() / (double) RAND_MAX) - rand() / (double) RAND_MAX) - 0.5);	#	= (nzero*0.9) * (1.0 +rand_coef * ole) rand() / (double) RAND_MAX) - 0.5))	
* (1. +rand_coef * (double) RAND_MAX) - 0.5)); +rand_coef * rand() / (double) RAND_MAX) - 0	9.13e		
+rand_coef * rand() / (double) RAND_MAX) - rand() / (double) RAND_MAX) - 0 self - diffusion */ rand_coef * rand_coef * (double) RAND_MAX) - 0 * (1.0 +rand_coef * / (double) RAND_MAX) - 0.5); * (1. +rand_coef * / (double) RAND_MAX) - 0.5); +rand_coef * rand() / (double) RAND_MAX) - rand() / (double) RAND_MAX) - 0.5);	*	= (nzero*0.9) * (1. +rand_coef * lble) rand() / (double) RAND_MAX) - 0.5))	
rand() / (double) RAND_MAX) - 0 self - diffusion */ rand_coef * rand() / (double) RAND_MAX) - 0 * (1.0 +rand_coef * / (double) RAND_MAX) - 0.5); * (1. +rand_coef * / (double) RAND_MAX) - 0.5); +rand_coef * rand() / (double) RAND_MAX) - rand()	*/ £10[k] £20[k]	0.0e+0; nzero * (1.0 +rand_coef * (((double) rand() / (double) RAND_MAX) -	
<pre>self - diffusion */ rand_coef * cand() / (double) RAND_MAX) - 0 * (1.0 +rand_coef * / (double) RAND_MAX) - 0.5); * (1. +rand_coef * / (double) RAND_MAX) - 0.5); +rand_coef * rand() / (double) RAND_MAX) -</pre>	/* f20[k] f10[k] */	0.0e+0; (nzero*0.9) * (1. +rand_coef * (((double) rand() / (double) RAND_MAX) -	
<pre>self - diffusion */ rand_coef * cand() / (double) RAND_MAX) - 0) * (1.0 +rand_coef * / (double) RAND_MAX) - 0.5); * (1. +rand_coef * +rand_coef * +rand_coef * +rand_coef * rand() / (double) RAND_MAX) - rand() / (double) RAND_MAX) -</pre>			
<pre>j=0; j< nnodes_y; j++) or(i=0; i < nnodes_x; i++) {</pre>	break; case 1: /* horizontal k=-1;	self - diffusion	
<pre>i < nnodes_x; i++) nnodes_x/2) nnodes_x/2) 10[k] = nzero * (1. +rand_coef * ((double) rand() / (double) RAND_MAX) - 0 20[k] = (nzero*0.9) * (1.0 +rand_coef * ((double) rand() / (double) RAND_MAX) - 0.5); * f(l(couble) rand() / (double) RAND_MAX) - 0.5); 20[k] = nzero * (1.0 +rand_coef * ((double) rand() / (double) RAND_MAX) - 0.5); 20[k] = nzero * (1.0 +rand_coef * ((double) rand() / (double) RAND_MAX) - 0.5); 20[k] = 0.00+0; </pre>	j=0; j<	s_y; j++)	
o * (1. +rand_coef * double) RAND_MAX) - 0 +0; zerc*0.9) * (1.0 +rand_coef * rand() / (double) RAND_MAX) - 0.5); zerc*0.9) * (1. +rand_coef * rand() / (double) RAND_MAX) - 0.5); co * (1.0 +rand_coef * (double) rand() / (doubl		nodes_x; i++)	
110[k] = nzero * (1. +rand_coef * (double) RAND_MAX) - 0 (double) rand() / (double) RAND_MAX) - 0 (20[k] = 0.00+0;	k++; if(i<=nnode	.s_x/2)	
<pre>f. (double) rand() / (double) RAND_MAX) - 0.5); // ((double) rand() / (double) RAND_MAX) - 0.5); /* f. (double) rand() / (double) RAND_MAX) - 0.5); f. (f. (double) rand() / (double) RAND_MAX) - 0.5); f. (f. (double) rand() / (double) RAND_MAX) - 0.6); f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX) - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (double) rand() / (double) RAND_MAX] - 0.60; f. (f. (f. (doub</pre>	(£10[k]	1	
* [10[k] = (nzero*0.9) * (1. +rand_coef * (double) rand() / (double) RAND_MAX) - 0.5); ((double) rand() / (double) rand(coef * (1.0 +rand_coef * (double) rand() / (double) RAND_MAX) - ((double) rand() / (double) RAND_MAX) - //	((c ((c () */	:] = (nzero*0.9) * (1.0 +rand_coef * iouble) rand() / (double) RAND_MAX) - 0.5));	
- 0.5)); NND_MAX) -	else {		
ı	/* £10[h] £20[k] £10[h] £20[k] £30[k]	- 0.5));	

Printed by sofonea from flumag3.mec.utt.ro

dif9init.c

Jun 20 1999 18:52

<pre>f10[k] = (nzero*0.1) * (1. +rand_coef *</pre>	
<pre>bbeak; case 8: /* horizontal diffusion couple, erf */ k=-1; for (j=0; j< nnodes_y; j++) for (i=0; i < nnodes_x; i++)</pre>	
<pre>f++; f10[k] = nzero * (1. +rand_coef * f10[k] = nzero * (1. +rand_coef *</pre>	
<pre>break; case 9: /* horizontal diffusion couple, linear */ k=-1; for (j=0; j< nnodes_y; j++) for (j=0; i < nnodes_x; i++)</pre>	
<pre></pre>	
/* f20[k] = (nzero*0.9) * (1.0 +rand_coef *	-
<pre>f10[k] = nzero * (1. +rand_coef *</pre>	
<pre>if(i>3*nnodes_x/4) { /* f10[k] = (nzero*0.9) * (1. +rand_coef *</pre>	
<pre>f10[k] = 0.0e+0; f20[k] = nzero * (1.0 +rand_coef *</pre>	
) break; case 2: /* vertical diffusion couple */ k=-1; for (j=0; j< nnodes_y; j++)	
for(1=0; i < nnodes_x; i++) {	

dif9init.c

Jun 20 1999 18:52	rage 5
<pre>k++; if()<=nnodes_y/2) f10[k] = nzero * (1. +rand_coef *</pre>	
<pre>110[k] = 0.0e+0; 120[k] = nzero * (1. +rand_coef * (((double) rand() / (double) RAND_MAX) -</pre>	0.5));
) break; case 5: /* horizontal diffusion couple with variable nleft, nright*/	
for (j=0; j< nnodes_y; j++) { for (i=0; i < nnodes_x; i++)	
{ k++; if(i<=nnodes_x/2)	
olleft * (1. +rand_coef * double) rand() / (double) RAND_MAX) - o2left * (1. +rand_coef * double) rand() / (double) RAND_MAX) -	0.5));
<pre>f10[k] = nzerolright * (1.0 +rand_coef *</pre>	0.5));
) break;)	
<pre>for (k=0; k < nnodes_all; k++)</pre>	
W W W W W W W W W W W W W W W W W W W	
<pre>fclose(frez);</pre>	
void init_arrays_nine_square_aux(void)	

<pre>int i, j, nnl1 = (nnodes_y-1)*nnodes_x; switch(key_boundary)</pre>
<pre>case 0: /* periodic boundaries - finite difference model */ for (i = 0; i< nnodes_all; i++) boundary_mode[i] = 0;</pre>
<pre>break; case 1: /* horizontal walls - finite difference model */ for (i = 0; i< nnodes_x; i++)</pre>
<pre>conndary_mode[1] = 1; for (i = nnodes_x; i < nnodes_all-nnodes_x; i++) boundary mode[i] = 0;</pre>
i = nnodes_all- dary_mode[i] =
<pre>case 2: /* vertical walls - finite difference model */ for (i = 0; i< nnodes_all; i++) boundary_mode(i) = 0; for (j=0; j < nnodes_y; j++)</pre>
{ boundary_mode[j*nnodes_x] = 3; boundary_mode[(j+1)*nnodes_x-1] = 4;
/* boundary_mode[j*nnodes_x+1] = 31; boundary_mode[(j+1)*nnodes_x-2] = 41; */
) brek; case 3: /* container - finite difference model */ for (= 0 : < nnodes all : i+)
Low (1 = 0; 1 × noveled 1 = 0; for (1 = 0; 4 × node 1 = 0; for (1 = 0; 4 × node 1 = 1 +)
<pre>boundary_mode[1] = 1; for (i = nnodes_all-nnodes_x; i < nnodes_all; i++) boundary_mode[i] = 2; for (i=0; i < nnodes_y; i++)</pre>
<pre>boundary_mode(j*nnodes_x] = 3; boundary_mode((j+1)*nnodes_x-1] = 4;</pre>
boundary_mode[0] = 5; boundary_mode[nnodes_x-1] = 6; boundary_mode[nnodes_x] = 7; boundary_mode[nnodes_x] = 7;
<pre>break; case 5, /* periodic boundaries - interpolation LGLB model */ for (i = 0; i< nnodes_all; i++) boundary_mode[i] = 10;</pre>
<pre>break; case 6: /* horizontal walls - finite difference model */ for (! = 0; i< nnodes_x; i++)</pre>
<pre>boundary_mode[1] = 11; for (1 = nnodes_x; i < nnodes_all-nnodes_x; i++) boundary_mode(x; i = 10.</pre>
<pre>fourdary_modetal = 12; four (1 = nnodes_all = nnodes_x; 1 < nnodes_all; 1++) boundary_mode[i] = 12;</pre>
<pre>break; case 7, /* vertical walls - interpolation LGLB model */ for (i = 0; i< nnodes_all; i++) boundary_mode[i] = 10; for (j=0; j < nnodes_y; j++)</pre>
boundary_mode[i*nnodes_x] = 13; boundary_mode[(j+1)*nnodes_x-1] = 14;
<pre>break; case 8: /* container - interpolation LGLB model */ for (1 = 0; / nonodes_all; i+) boundary_mode[i] = 10;</pre>

_	1
0	1
_	1
=	ı
=	ı
	ı
ŭ	ı
₩.	ı
=	ı
agg.mec.utt.ro	ı
· .	Į
O	1
Œ	1
ë	1
<	1
$\overline{}$	1
=	
_	
⊏	ı
0	
=	
_	
Œ	ı
w.	ı
_	ı
d by sotonea from fluma	
≒	1
\mathbf{x}	ľ
υ,	ł
➣	3
0	
_	1
Printec	
உ	
ć	
=	
~	
_	

**********	****************	************************************	**
300			* * *
O*************	化水杨素 化 化 化 化 化 化 化 化 化 化 化 化 化 化 化 化 化 化 化	<u> </u>	* * *
<pre>#include <stdlib.h> #include <math.h> #include <stdio.h></stdio.h></math.h></stdlib.h></pre>	b.h> h> .h>		
#include "dif9head.h"	ead, h"		
oid init_latt	void init_lattice_functions(void)		
11		izeof (double));	
fil = (double*		izeof(double));	
fl3 = (double* f14 = (double*		izeof(double)); izeof(double));	
11			
f16 = (double* f17 = (double*		<pre>izeo1(double)); izeof(double));</pre>	
$f20 = (double^*)$		izeoi(double);	
11			
f23 = (double*	*) malloc(nnodes_all*sizeof *) malloc(nnodes_all*sizeof	izeof(double));	
	-		
f28 = (double*	_	izeof (double));	
	£ 3	sizeof (double);	
ffll = (double*	Le*) malloc(nnodes_all*: e*) malloc(nnodes all*:	sizeoi(aoubie)); sizeof(double));	
1 11	e*) malloc(nnodes_all*;	sizeof(double));	
ff14 = (double*	le*) malloc(nnodes_all*;	sizeof(double));	
ff16 = (double*	.e*) malloc(nnodes_all*; .e*) malloc(nnodes_all*;	sizeof(double);	
	le*) malloc(nnodes_all*sizeof	sizeof(double));	
11	-		
11 11	*) malloc(nnodes;	_all*sizeof(double)); all*sizeof(double));	
	*) malloc(nnodes,		
11) malloc(nnodes	sizeof(double));	
ff27 = (double*	malloc(nnodes)		
#) malloc(nnodes		
sfl0 = (double*) malloc(nnodes	_all*sizeof(double); all*sizeof(double));	
	_		
sfl3 = (double*	<pre>le*) malloc(nnodes_all*sizeof le*) malloc(nnodes all*sizeof</pre>	<pre>sizeof(double)); sizeof(double));</pre>	
1	_		
) = -	le*) malloc(nnodes_all*sizeof	Sizeof(double);	
sii/ = (double sii) = (double sii)	(* 0 * 0		
		sizeof (double);	
sf21 = (double*	Le*) malloc(nnodes_all*sizeot o*) malloc(nnodes_all*sizeof	sizeor(double));	
f23 = (-		
= 4		sizeof(double);	
sf25 = (double*	le*) malloc(nnodes_all*sizeor e*) malloc(nnodes_al]*sizeof	sizeor (double);	
	_	sizeof(double);	
೭	=	sizeof(double);	
	[a a a a a a a a a a	*81480+(3010)10111	

	10000000		1252
regil = (double) malloc (innodes_all = sizec (double)) regil = (fil) free (fil) reg		Z. All Saus.	Lager
	- 1	[display 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	
media (couble) malloc (nodes_all*size(couble)) media (couble) malloc (no	01	malloc(nnodes_all*sizeoi(doubl	
### ### ### ### ### ### ### ### ### ##	11	malloc(nnodes_ail*sizeoi	
negis = (Gonbles) malloc (Innodes_all'sizeof (Gonble)) negis = (Gonbles) malloc (Inno	į	malloc(nnodes_all*sizeof	
medic = ((couble *) malloc ((nodes_all*size) ((couble)) muxloc = ((couble *) malloc ((nodes_all*size) ((couble)) muxloc = ((couble *) malloc ((nodes_all*size) ((couble)) muxloc = ((couble *) malloc ((nodes_all*size) ((int)) muxloc = ((couble *) malloc ((nodes_all*	Į	malloc(nnodes_all*sizeof	
	11	malloc(nnodes_all*sizeof	
medit = (couble =) malloc (modes_all'sizeof (double)) mid = (double =) malloc (modes_all'sizeof (double)) mid = (double =) malloc (modes_all'sizeof (double)) mid = (double =) malloc (modes_all'sizeof (int)) mid = (double =) malloc (modes_all'sizeof (double)) mid = (double =) malloc (modes_all'sizeof (int)) mid = (mid =) mid = (mid =) mid = (mid =) mid = (m		malloc(nnodes_all*sizeof	
neg70 = (double*) malloc(nnodes_all*sized(double)) neg70 = (double*) malloc(nnodes_all*sized(double)) neg71 = (double*) malloc(nnodes_all*sized(double)) neg72 = (double*) malloc(nnodes_all*sized(double)) neg73 = (double*) malloc(nnodes_all*sized(double)) neg74 = (double*) malloc(nnodes_all*sized(double)) neg75 = (double*) malloc(nnodes_all*sized(double)) neg75 = (double*) malloc(nnodes_all*sized(double)) neg76 = (double*) malloc(nnodes_all*sized(double)) neg77 = (double*) malloc(nnodes_all*sized(double)) neg78 = (double*) malloc(nnodes_all*sized(double)) neg70 = (double*) malloc(nnodes_all*sized(double)) neg70 = (double*) malloc(nnodes_all*sized(double)) neg70 = (double*) malloc(nnodes_all*sized(double)) neg70 = (double*) malloc(nnodes_all*sized(double)) neg71 = (int*) malloc(nnodes_all*sized(int)) neg72 = (int*) malloc(nnodes_all*sized(int)) neg73 = (int*) malloc(nnodes_all*sized(int)) neg72 = (int*) malloc(nnodes_all*sized(int)) neg72 = (int*) malloc(intodes_all*sized(int)) neg72 = (int*) malloc(intodes_all*sized(int)) neg72 = (int*) malloc(intodes_all*sized(int)) neg72 = (int*) malloc(intodes_all*sized(int)) neg73 = (int*) malloc(intodes_all*sized(int)) neg74 = (int*) malloc(intodes_all*sized(int)) neg75 = (int*) malloc(intodes_all*sized(int)) neg75 = (int*) malloc(intodes_all*sized(int) neg75 = (int*) malloc(intodes_all*si	II	malloc(nnodes_all*sizeof	
heg2 = (double*) malloc(nnodes_all*sizeof(double); heg2 = (double*) malloc(nnodes_all*sizeof(double); heg2 = (double*) malloc(nnodes_all*sizeof(double); heg3 = (double*) malloc(nnodes_all*sizeof(double); wilco	11	malloc(nnodes_all*sizeof	
head2 = (double*) malloc(unodes_all*sizeof(double)) head2 = (double*) malloc(unodes_all*sizeof(double)) head3 = (double*) malloc(unodes_all*sizeof(double)) head4 = (double*) malloc(unodes_all*sizeof(double)) head5 = (double*) malloc(unodes_all*sizeof(duuble)) head5 = (double*) malloc(unodes_all*sizeof(unol)) free(fil)) free(fil)) free(fil) free(f	II	malloc(nnodes_all*sizeof	
neg2 = (double*) malloc (mnodes_all*sizeof (double)) neg3 = (double*) malloc (mnodes_all*sizeof (double)) neg5 = (double*) malloc (mnodes_all*sizeof (double)) neg5 = (double*) malloc (mnodes_all*sizeof (double)) neg6 = (double*) malloc (mnodes_all*sizeof (double)) neg7 = (double*) malloc (mnodes_all*sizeof (double)) neg8 = (double*) malloc (mnodes_all*sizeof (double)) neg8 = (double*) malloc (mnodes_all*sizeof (double)) neg8 = (int*) malloc (mnodes_all*sizeof (int)) free (int)	11	malloc(nnodes_all*sizeof	
heg4 = (double*) malloc(nnodes_all*sizeof(double); heg4 = (double*) malloc(nnodes_all*sizeof(double); heg5 = (double*) malloc(nnodes_all*sizeof(double); heg6 = (double*) malloc(nnodes_all*sizeof(double); heg8 = (double*) malloc(nnodes_all*sizeof(double); hug7 = (double*) malloc(nnodes_all*sizeof(double); hug2 = (double*) malloc(nnodes_all*sizeof(double); hug2 = (double*) malloc(nnodes_all*sizeof(double); hug1 = (int*) malloc(nnodes_all*sizeof(int); hug2 = (int*) malloc(nnodes_all*sizeof(int); hug1 = (int*) malloc(nnodes_all*sizeof(int); hug2 = (int*) malloc(nnodes_all*sizeof(int); hug4 = (int*) malloc(nnodes_all*size	11	malloc(nnodes_all*sizeof	
neg5 = (double*) malloc (modes_all'sizeof (double)); neg5 = (double*) malloc (modes_all'sizeof (double)); neg5 = (double*) malloc (modes_all'sizeof (double)); neg7 = (double*) malloc (modes_all'sizeof (double)); uvioc = (double*) malloc (modes_all'sizeof (double))	11) malloc(nnodes_all*sizeof	
neg56 = (double*) malloc (modes_all'sized (double)); neg76 = (double*) malloc (modes_all'sized (double)); neg78 = (double*) malloc (modes_all'sized (double)); neg78 = (double*) malloc (modes_all'sized (double)); uvloc = (double*) malloc (modes_all'sized (doubl	1	malloc(nnodes_all*sizeof	
neg2 = (double*) malloc(innodes_all*sizeof(double)); neg2 = (double*) malloc(innodes_all*sizeof(double)); uvloc = (double*) malloc(innodes_all*sizeof(duuble)); uvloc = (double*) malloc(innodes_all*sizeof(int)); uvloc = (duuble*) malloc(innodes		malloc(nnodes_all*sizeof	
neg78 = (double*) malloc(innodes_all*sizeof(double)); uvloc = (double*) malloc(innodes_all*sizeof(inn)); uvloc = (double*) malloc(innodes_all*sizeof	-) malloc(nnodes_all*sizeof	
<pre>neqse = (double*) malloc (innodes_all*sizeof (double)); uvloc = (double*) malloc (innodes_all*sizeof (double)); uvloc = (double*) malloc (innodes_all*sizeof (double)); uvloc1 = (double*) malloc (innodes_all*sizeof (double)); uvloc2 = (double*) malloc (innodes_all*sizeof (double)); uvloc2 = (double*) malloc (innodes_all*sizeof (duuble)); uvloc2 = (double*) malloc (innodes_all*sizeof (duuble)); uvloc3 = (int*) malloc (innodes_all*sizeof (int)); uvloc4 = (int*) malloc (innodes_all*sizeof (int)); uvloc4 = (int*) malloc (innodes_all*sizeof (int)); uvloc5 = (int*) malloc (innodes_all*sizeof (int)); uvloc6 = (int*) malloc (int); uvloc6 = (int) malloc (int); uv</pre>	1	malloc(nnodes all*sizeof	
<pre>wile = (double*) mailor(innodes_all*sizeof(double)); wile = (double*) mailor(innodes_all*sizeof(duib)); wile = (double*) mailor(innodes_all*sizeof(double)); wile = (double*) mailor(innodes_all*sizeof(double)); wile = (double*) mailor(innodes_all*sizeof(double)); wile = (double*) mailor(innodes_all*sizeof(double)); free(fils); fr</pre>	1	malloc(nnodes all*sizeof	
<pre>vyloc = (double*) malloc(innodes_all*sizeof(double)); vyloc1 = (double*) malloc(innodes_all*sizeof(double)); vyloc1 = (double*) malloc(innodes_all*sizeof(double)); vyloc2 = (double*) malloc(innodes_all*sizeof(duble)); vyloc2 = (double*) malloc(innodes_all*sizeof(duble)); vyloc3 = (double*) malloc(innodes_all*sizeof(duble)); vyloc3 = (duble*) malloc(innodes_all*sizeof(duble)); vyloc3 = (duble*) malloc(innodes_all*sizeof(duble)); vyloc4 = (duble*) malloc(innodes_all*sizeof(duble)); vyloc5 = (duble*) malloc(innodes_all*sizeof(duble)); vyloc6 = (duble*) malloc(innodes_all*sizeof(duble)); vyloc6 = (duble*) malloc(innodes_all*sizeof(duble)); vyloc7 = (double*) malloc(innodes_all*sizeof(duble)); vyloc8 = (double*) malloc(innodes_all*sizeof(duble)); free(fil); fr</pre>	-		
wided = (double*) malloc(innodes_all*sizeof(double)); wiloc2 = (double*) malloc(innodes_all*sizeof(double)); wiloc2 = (double*) malloc(innodes_all*sizeof(double)); wiloc2 = (double*) malloc(innodes_all*sizeof(duble)); wiloc3 = (double*) malloc(innodes_all*sizeof(lut)); wiloc3 = (dibta; malloc(innodes_all*sizeof(lut)); wiloc4 = (dibta; malloc(innodes_all*sizeof(lut)); wiloc4 = (dibta; malloc(innodes_all*sizeof(lut)); wiloc5 = (double*) malloc(innodes_all*sizeof(lut)); wiloc6 = (dibta; malloc(innodes_all*sizeof(lut)); wiloc6 = (double*) malloc(innodes_all*sizeof(lut)); wiloc6 = (double*) malloc(innodes_all*sizeof(lut)); free(fil);			
<pre>uvloc2 = (double*) malloc(innodes_all*sizeog(double)); uvloc2 = (double*) malloc(innodes_all*sizeog(double)); uvloc2 = (double*) malloc(innodes_all*sizeog(double)); uvloc2 = (double*) malloc(innodes_all*sizeog(duble)); uvloc3 = (int*) malloc(innodes_all*sizeof(int)); uvloc4 = (int*) malloc(innodes_all*sizeof(int)); uvloc5 = (double*) malloc(innodes_all*sizeof(int)); uvloc6 = (double*) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc8 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc8 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc8 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc8 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc6 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc6 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*sizeof(int)); uvloc7 = (uvloc6) malloc(innodes_all*si</pre>	1	foesia*[[e sepondo]]ocia*[[em (
UNIOCI = ((ouble*) malloc (unodes_all*sizeof(double)); UNIOC2 = ((ouble*) malloc (unodes_all*sizeof(double)); UNIOC2 = ((ouble*) malloc (unodes_all*sizeof(double)); UNI = ((int*) malloc (unodes_all*sizeof(int)); UNI =	1	marroc (minutes arrested)	
<pre>uulcoz = (double*) malloc (innodes_all*sizeof (double)); vulcoz = (double*) malloc (innodes_all*sizeof (double)); vul = (int*) malloc (innodes_all*sizeof (int)); free (int*) malloc (int*) malloc (int*) free (int*) fre</pre>	(go 	malloc(nhodes_all*sizeoi	
<pre>vyloc2 = (double*) malloc(innodes_all*sizeof (double)); nv1 = (int*) malloc(innodes_all*sizeof (duble)); nv3 = (int*) malloc(innodes_all*sizeof(int)); nv3 = (int*) malloc(innodes_all*sizeof(int)); nv4 = (int*) malloc(innodes_all*sizeof(int)); nv5 = (int*) malloc(innodes_all*sizeof(int)); nv6 = (int*) malloc(innodes_all*sizeof(int)); nv7 = (int*) malloc(innodes_all*sizeof(int)); nv8 = (int*) malloc(innodes_all*sizeof(int)); nv9 = (int*) malloc(innodes_all*sizeof(iouble)); nloc1 = (double*) malloc(innodes_all*sizeof(iouble)); nloc2 = (double*) malloc(innodes_all*sizeof(iouble)); free(fil); free(</pre>) 미	malloc(nnodes_all*sizeoi	
nv2 = (int*) malloc(modes_all*sizeof(int)); nv3 = (int*) malloc(modes_all*sizeof(int)); nv4 = (int*) malloc(modes_all*sizeof(int)); nv5 = (int*) malloc(modes_all*sizeof(int)); nv6 = (int*) malloc(modes_all*sizeof(int)); nv7 = (int*) malloc(modes_all*sizeof(int)); nv8 = (int*) malloc(modes_all*sizeof(int)); nv8 = (int*) malloc(modes_all*sizeof(int)); nv8 = (int*) malloc(modes_all*sizeof(double)); nloc1 = (double*) malloc(modes_all*sizeof(double)); nloc2 = (double*) malloc(modes_all*sizeof(double)); free(f10); free(f21);) 의	malloc(nnodes_all*sizeof	
nv2 = (int*) malloc (innodes_all*sizeof(int)); nv4 = (int*) malloc (innodes_all*sizeof(int)); nv5 = (int*) malloc (innodes_all*sizeof(int)); nv6 = (int*) malloc (innodes_all*sizeof(int)); nv7 = (int*) malloc (innodes_all*sizeof(int)); nv8 = (int*) malloc (innodes_all*sizeof(int)); free(fil);	(int*)	c(nnodes_all*sizeof(int))	
<pre>nv3 = (int*) malloc(innodes_all*sizeof(int)); nv4 = (int*) malloc(innodes_all*sizeof(int)); nv5 = (int*) malloc(innodes_all*sizeof(int)); nv6 = (int*) malloc(innodes_all*sizeof(int)); nv8 = (int*) malloc(innodes_all*sizeof(int)); nv8 = (int*) malloc(innodes_all*sizeof(int)); nv9 = (int*) malloc(innodes_all*sizeof(int)); boundary_mode = (int*) malloc(innodes_all*sizeof(int)); nloc1 = (double*) malloc(innodes_all*sizeof(int)); free(fil); free(f</pre>	= (int*)	alloc(nnodes all*sizeof(int));	
<pre>nv4 = (int*) malloc(nnodes_all*sizeof(int)) nv5 = (int*) malloc(nnodes_all*sizeof(int)) nv6 = (int*) malloc(nnodes_all*sizeof(int)) nv7 = (int*) malloc(nnodes_all*sizeof(int)) nv8 = (int*) malloc(nnodes_all*sizeof(int)); nv8 = (int*) malloc(nnodes_all*sizeof(int)); nloc1 = (double*) malloc(nnodes_all*sizeof(double)); nloc2 = (double*) malloc(nnodes_all*sizeof(double)); nloc2 = (double*) malloc(nnodes_all*sizeof(double)); nloc2 = (double*) malloc(nnodes_all*sizeof(double)); free(fil); f</pre>	(4-trt-)	2]]Or (nnodes 2] *aiveof (int) :	
	1	man of the contract of the con	
<pre>nvp = (int*) malloc(innodes_all*sizeof(int)); nvf = (int*) malloc(innodes_all*sizeof(int)); nvg = (int*) malloc(innodes_all*sizeof(int)); nvg = (int*) malloc(innodes_all*sizeof(int)); nvg = (int*) malloc(innodes_all*sizeof(int)); nloc1 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); free(fil); free(fil);</pre>	(1117)	ALLOC CALCOLOR STATE STATE OF THE CALCULATION OF TH	
<pre>nv6 = (int*) malloc(nnodes_all*sizeof(int)); nv7 = (int*) malloc(nnodes_all*sizeof(int)); boundary_mode = (int*) malloc(nnodes_all*sizeof(int)); boundary_mode = (int*) malloc(nnodes_all*sizeof(double)); nloc1 = (iouble*) malloc(nnodes_all*sizeof(double)); nloc2 = (double*) malloc(nnodes_all*sizeof(double)); loc2 = (double*) malloc(nnodes_all*sizeof(double)); free(fil); free(fil);</pre>	= (int*)	alloc(nnodes_all*sizeof(int));	
<pre>nv7 = (int*) malloc(incdes_all*sizeof(int)); nv8 = (int*) malloc(incdes_all*sizeof(int)); nloc1 = (iouble*) malloc(incdes_all*sizeof(int)); nloc2 = (iouble*) malloc(incdes_all*sizeof(double)); nloc2 = (iouble*) malloc(incdes_all*sizeof(double)); nloc2 = (iouble*) malloc(incdes_all*sizeof(double)); free(fil); free</pre>	= (int*)	alloc(nnodes_all*sizeof(int));	
<pre>nv8 = (int*) malloc(innodes_all*sizeof(int)); boundary_mode = (int*) malloc(innodes_all*sizeof(int)); nloc1 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); free(fil); free(fil);</pre>	= (int*)	alloc(nnodes_all*sizeof(int));	
<pre>boundary_mode = (int*) malloc(nnodes_all*sizeof(int)); nloo1 = (double*) malloc(nnodes_all*sizeof(double)); nloo2 = (double*) malloc(nnodes_all*sizeof(double)); nloo2 = (double*) malloc(nnodes_all*sizeof(double)); free(fil); free(fil);</pre>	= (int*)	alloc(nnodes all*sizeof(int));	
Dict = (double*) malloc(innodes_all*sizeof(double)); nloc1 = (double*) malloc(innodes_all*sizeof(double)); nloc2 = (double*) malloc(innodes_all*sizeof(double)); free (fil);	Agents mode	" ('n+*) ma]] 00 (nnode a] *e; anof ('n+') .	
	į,	- (IIIC) MARKOC (IIIOCCO THE BIRGOCE (IIIC))	
void free_lattice_functions (void)	1 1	e.) mailoc(modes all*sizeof(domble)),	
<pre>void free_lattice_functions(void) { free(fil); free(f</pre>			
<pre> {</pre>			
free(f10); free(f11); free(f12); free(f12); free(f13); free(f13); free(f15); free(f2); free(f13); free(f2); free(f2); free(f2);	void free_lattice	e_functions (void)	
free(f11); free(f12); free(f13); free(f14); free(f13); free(f15); free(f21); free(f21); free(f23); free(f23); free(f24); free(f24); free(f21);	-		
free(fil);	free(f10);		
free(fil2); free(fil4); free(fil5); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil1); free(fil1); free(fil1); free(fil1); free(fil2); free(fil2); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8); free(fil8);	free (f11);		
free(f13); free(f13); free(f15); free(f15); free(f21); free(f11); free(f11); free(f11); free(f11); free(f11); free(f21);	(£12)		
free(f14); free(f15); free(f17); free(f21); free(f22);	(£13)		
free(f15); free(f18); free(f21); free(f22); free(f23); free(f23); free(f23); free(f21); free(f21); free(f21); free(f21); free(f13); free(f13); free(f13); free(f13); free(f13); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21);	(£14)		
free(f16); free(f18); free(f21); free(f21); free(f21); free(f22); free(f23); free(f24); free(f25); free(f25); free(f21); free(f21); free(f13); free(f13); free(f13); free(f13); free(f13); free(f13); free(f13); free(f21); free(f21); free(f22); free(f22); free(f22);	(£15)		
free(f18); free(f21); free(f22); free(f23); free(f23); free(f23); free(f23); free(f21); free(f21); free(f21); free(f11); free(f12); free(f13); free(f13); free(f13); free(f13); free(f13); free(f21); free(f22); free(f22); free(f22);	£15)		
free(f20); free(f21); free(f23); free(f23); free(f23); free(f23); free(f23); free(f21); free(f21); free(f21); free(f11); free(f11); free(f11); free(f11); free(f11); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21); free(f22); free(f22);	1110		
free(f20); free(f21); free(f23); free(f23); free(f24); free(f27); free(f27); free(f27); free(f27); free(f13); free(f13); free(f13); free(f13); free(f13); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21); free(f21);	100		
<pre>free(f21); free(f21); free(f23); free(f24); free(f25); free(f25); free(f21); free(f11); free(f11); free(f11); free(f11); free(f11); free(f12); free(f21); free(f21);</pre>	TTR)		
free(f21); free(f23); free(f24); free(f24); free(f25); free(f27); free(f27); free(f110); free(f113); free(f113); free(f113); free(f113); free(f113); free(f113); free(f113); free(f113); free(f113); free(f21); free(f21); free(f21);	£20)		
free(f22); free(f23); free(f24); free(f25); free(f27); free(f11); free(f11); free(f11); free(f11); free(f12); free(f12); free(f12); free(f21); free(f21); free(f21);	£21)		
free(f23); free(f25); free(f25); free(f21); free(f21); free(f11); free(f13); free(f13); free(f13); free(f13); free(f13); free(f21); free(f21); free(f21);	£22)		
free(f24); free(f25); free(f27); free(f27); free(f11); free(f11); free(f13); free(f13); free(f13); free(f13); free(f21); free(f21); free(f21);	£23)		
free(f25); free(f27); free(f27); free(f11); free(f11); free(f113); free(f13); free(f13); free(f13); free(f20); free(f20); free(f21);	free (f24);		
free (f26); free (f27); free (f27); free (f11); free (f21); free (f21); free (f21); free (f21); free (f21);	frop(f25):		
free((23); free(f23); free(f11); free(f11); free(f13); free(f15); free(f15); free(f18); free(f20); free(f21);	(000) (400)		
free (f28); free (f110); free (f111); free (f113); free (f113); free (f113); free (f113); free (f113); free (f110); free (f110); free (f110); free (f121); free (f21); free (f21);	free (f27).		
free(fil); free(fil); free(fil); free(fil); free(fil); free(fil); free(fil); free(fil); free(fil); free(fil);	froo (£28) •		
free(fil);	1 (071) 3217		
<pre>free(fill); free(fill); free(fill);</pre>	ree(IIIO);		
<pre>free(ff13; free(ff14); free(ff15); free(ff15); free(ff18); free(ff18); free(ff20); free(ff21); free(ff21);</pre>	ree(IIII);		
<pre>free(fil3); free(fil3); free(fil5); free(fil7); free(fil3); free(fil3); free(fil3); free(fil3); free(fil3); free(fil3);</pre>	free(ff12);		
<pre>free(f11); free(f11); free(f11); free(f11); free(f20); free(f20); free(f21); free(f21);</pre>	free (ff13);		
<pre>free(f15); free(f17); free(f17); free(f20); free(f21); free(f21); free(f22);</pre>	free (ff14);		
<pre>free(ff16); free(ff17); free(ff18); free(ff20); free(ff21); free(ff21); free(ff21);</pre>	free (ff15):		
free(ff1); free(ff18); free(ff21); free(ff21); free(ff21);	from (ff16) :		
<pre>free(f18); free(f20); free(f21); free(f22); free(f22);</pre>	1001110011		
<pre>ree(f(20); free(f(21); free(f(21); free(f(22); free(f(22);</pre>	(/TTT) AATT		
<pre>free(ff21); free(ff21); free(ff22); free(ff23);</pre>	rree(rrrs);		
<pre>free(ff21); free(ff22); free(ff23);</pre>	iree(iiZU);		
free(ff22); free(ff23):	free(ff21);		
free(ff23);	free(ff22);		
	free (ff23);		

Mar 30 1999 08:42	dif9aux.c	Page 4
{ if(nf0[k] < 0.00)		
<pre>fprintf(frez,"\n\nNEGAIIVE iter,k,nf0[k]); exit(1); }</pre>	\nNEGAIIVE F: iter=%d k=%d nf0=%g\n", £0[k]);	
	<pre>\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\</pre>	
<pre>{ fprint(frez, "\n\nNsGAIIVE iter, k, nf2[k]); exit(1); } if(nf3[k] < 0.00)</pre>	\nNEGATIVE F: iter=%d k=%d nf2=%g\n", f2[k]);	
<pre>fprintf(frez,"\n\nNEGAIIVE F:</pre>	\nNEGATIVE F: iter=%d k=%d nf3=%g\n", f3[k]);	
<pre>{</pre>	\nNEGATIVE F: iter=%d k=%d nf4=%g\n", f4[k]);	
<pre>{</pre>	\nNEGRIVE F: iter=%d k=%d nf5=%g\n", f5[k]);	
<pre>fprintf(frez,"\n\nNEGATIVE F: exit(1); if(nf7[k] < 0.00)</pre>	\nNEGAIIVE F: iter=%d k=%d nf6=%g\n", f6[k]);	
<pre></pre>	<pre>fprintf(frez,"\n\nNBGAIVE F: iter=%d k=%d nf7=%g\n", exit(1); f8[k] < 0.00)</pre>	
<pre>fprintf(frez,"\n</pre>	<pre>fprintf(frez,"\n\nNBGAIIVE F: iter=%d k=%d nf8=%g\n", exit(1); ter,k,nf8[k]); cez);</pre>	
<pre>void store_distribution_functions(void) { int k; for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>	<pre>iunctions(void) k++)</pre>	
{ sf10[k] = f10[k]; sf11[k] = f11[k]; sf12[k] = f12[k]; sf13[k] = f13[k]; sf14[k] = f14[k];		

Apr 12 1999 18:42 dif9draw.c	Page 1
/*************************************	
<pre>#include <stdio.h> #include <math.h> #include <stdlib.h> #include <string.h></string.h></stdlib.h></math.h></stdio.h></pre>	
<pre>#include "dif9head.h" /* you may modify gray_levels */ int gray levels = 63, base = 35;</pre>	
<pre>void xv_new(double n0[], double n2[], double n3[], double n4[], double n5[], double n6[], double n7[], double n8[], char arg_name[], double maxval)</pre>	
int i, val; char xv_name[128]; FILE *fxv;	
<pre>sprintf(xv_name, "%s.%06d.xpm", arg_name, iter); fxv = fopen(xv_name, "w"); fprintf(fxv, "",* XPM *\\nstatch char * init_xpm[] = {\\n"}; fprintf(fxv, "\",* XPM *\\nstatch char * init_xpm[] = {\\n"}; fprintf(fxv, "\",* XPM *\\nstatch char * init_xpm[] = {\\n"}; fprintf(fxv, "\",* XPM *\\nstatch char * init_xpm[] = {\\n"}; for(1 = 0; i = gray_lavels; i+1, fprintf(fxv, "\","\\n\",* \\n\",* \\n'"," \\n'",* \\n</pre>	
<pre>for(1 = 0; i <= nnodes_all; i++)</pre>	
<pre>if(1 == (i+1) % nnodes_x)) fprintf(fxv, "\", \n\""); val = floor ((n0[i] + n1[i] + n2[i] + n3[i] + n4[i] + n5[i] + if(val > gray_levels) val = gray_levels; if(val < 0) val = 0; val = 0; f(val < 0) val = 0; f(val < 0)</pre>	
fp!intf(fxv,"\"); fclose(fxv); }	
<pre>void xv(double n0[], double n1[], double n2[], double n3[], double n5[], double n6[], double n7[], double n8[], char arg_name[])</pre>	

Page 2 FILE *pntot, *prhotot, *prho1, *prho2, *pomegal, *pomeganl, *pomegacl, *pomeganl, *pj2, *pul, *pu2, *pu, *pu2bls, *publs; val = (floor((fil[k]*ecx[i]+fl2[k]*ecx[2]+fl3[k]*ecx[3]+ fl4[k]*ecx[4]+fl5[k]*ecx[5]+fl6[k]*ecx[6]+ fl7[k]*ecx[7]+fl8[k]*ecx[8])*l000.)); dif9draw.c sprintf(xv_name,"%s.%05dP",arg_name,iter); fxv = fopen(xv_name,"wt"); fprintf(fxv,"P2)n%3d%4d\n63\n",nnlun,nnlat); for(K=1; k<=nnod; k++)</pre> char ntot_profile_name[128]; char chotot_profile_name[128]; char chol_profile_name[128]; char chol_profile_name[128]; char omegal_profile_name[128]; char omegal_profile_name[128]; char omegal_profile_name[128]; char i2_profile_name[128]; char i2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2_profile_name[128]; char u2_brofile_name[128]; char u2_brofile_name[128]; char u2_brofile_name[128]; val = 63; if(val < 0) val = 0; val = 63 - val; */ fprintf(fxv, *83d", val); if(i == 15)</pre> fprintf(fxv, "%3d", val); if(i == 15) fprintf(fxv,"\n"); } fprintf(fxv,"\n"); void dif9_profile (void) if(i) fprintf(fxv, "\n"); if(val > 63) val = 63; if(val < 0) val = 0; val = 63 - val;</pre> fprintf(fxv,"\n"); fclose(fxv); Apr 12 1999 18:42 fclose(fxv); *

Printed by sofonea from flumag3.mec.utt.ro

2 dif9draw.c

int i, j, k, kk, nval; dodolo, rholoc1, rholoc2, rholoc, omegalloc, omegalloc, omegalloc, omegalloc, uloc1, uloc2, jloc2, uloc; double uloc1bis, uloc2bis; ulocbis;

val = floor(n0[k]+n1[k]+n2[k]+n3[k]+n4[k]+n5[k]+nn[k]+nn2[k]+nn2[k]+nn2[k])*63.0);

if(val > 63)

File .txv;
Int i, k, val;
int i, k, val;
sprint[(xv_name, "%s.%05d", arg_name, iter);
fprint((xv_name, "wt");
fprintf(fxv, "P2\n%3d%4d\n63\n", nnodes_x, nnodes_y);
for (k=0; k<nnodes_all; k++)</pre>

FILE *fxv;

Apr 12	Apr 12 1999 18:42 dif9draw.c	Page 3
double ul, u	double rho1, rho2, rho, omega1, omegan1, omega2, omegan2, u1, u2, j2, u, u2b1s, ubis; double ntot, rhotot;	
Sprin Sprin Sprin Sprin Sprin Sprin Sprin Sprin Sprin Sprin	<pre>sprintf(ntot_profile_name,"PNTOTAL%s.%06d",id_name,iter); sprintf(rhotot_profile_name,"PRHOTAL%s.%06d",id_name,iter); sprintf(rhotot_profile_name,"PRHO1%s.%06d",id_name,iter); sprintf(rhot_profile_name,"PRHO1%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGAMASS1%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGANA%s.%06d",id_name,iter); sprintf(omegal_profile_name,"POMEGANA%s.%06d",id_name,iter); sprintf(ul_profile_name,"PUS*s.%06d",id_name,iter); sprintf(ul_profile_name,"PUS*s.%06d",id_name,iter); sprintf(ul_profile_name,"PUS*s.%06d",id_name,iter); sprintf(ul_profile_name,"PUS*s.%06d",id_name,iter); sprintf(ul_profile_name,"PUS*s.%06d",id_name,iter); sprintf(ul_profile_name,"PUSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS</pre>	
pntot = printot = printot = printot = printol = i prin	<pre>t = fopen(ntot_profile_name,"wt"); t = fopen(rhotot_profile_name,"wt"); t = fopen(rhoto_profile_name,"wt"); 2 = fopen(rhot_profile_name,"wt"); 2 = fopen(comegal_profile_name,"wt"); gan1 = fopen(comegal_profile_name,"wt"); gan2 = fopen(comegal_profile_name,"wt"); gan2 = fopen(comegal_profile_name,"wt"); elopen(log_log_log_log_log_log_log_log_log_log_</pre>	
pu2bis = pubis = */ for (1=0;	<pre>pulbis = fopen(ulbis_profile_name,"wt"); pubis = fopen(ubis_profile_name,"wt"); // for(!=0: i<nnodes i++)<="" pre="" x:=""></nnodes></pre>	
¥ 4 4 4 0 0 0 0 113 3	00000	
	u2bis = 0.0000; u2bis = 0.0000; ubis = 0.0000; f_r(j=0; j <nnodes_y; j++)<="" td=""><td></td></nnodes_y;>	
	<pre>' k = 1 + j * nnodes_x; nloc1 = filo[k]+filok]+filok]+filok]+filok]+filok]+filok]+ filok]+filok</pre>	
	<pre>printf("k=%d nloc1=%g nloc2=%g\n",k,nloc1,nloc2); */ rholoc1 = mass1 * nloc1; rholoc2 = mass2 * nloc2; rholoc = rholoc1 + rholoc2;</pre>	
	/* printf("k=%d rholoc1=%g rholoc2=%g\n", k,rholoc1,rholoc2,rholoc); */ if(rholoc)	

Apr 12 19	1999 18:42	dif9draw.c	Page 4
	{ omegalloc = rholoc1 / rh omegaloc = rholoc2 / rh	rholoc; rholoc;	
	else { omegalloc = 0.0000;		
	D I 46	ouos; rholoc=%g\n",k,rholoc);	
	32)	;	
	omeganlloc = nloc1 / (nl omeganlloc = nloc2 / (nl } else	(nloc1+nloc2); (nloc1+nloc2);	
	{ omegan11oc = 0.0000; omegan21oc = 0.0000; }		
	<pre>if(nloc1) uloc1 = (f11[k]*ecx1[1]- f14[k]*ecx1[4]- f17[k]*ecx1[7]- else uloc1 = 0.0000;</pre>	<pre>(fil[k]*ecx1[1]+fil[k]*ecx1[2]+fil3[k]*ecx1[3]+ fil4[k]*ecx1[4]+fil5[k]*ecx1[5]+fil6[k]*ecx1[6]+ fil7[k]*ecx1[7]+fil8[k]*ecx1[8]) / nloc1; 0.0000;</pre>	
	<pre>if(nloc2) uloc2 = (f21[k]*ecx2[1]- f24[k]*ecx2[4]- f27[k]*ecx2[7]- else uloc2 = 0.0000;</pre>	(f21[k]*ecx2[1]+f22[k]*ecx2[2]+f23[k]*ecx2[3]+ f24[k]*ecx2[4]+f25[k]*ecx2[5]+f26[k]*ecx2[6]+ f27[k]*ecx2[7]+f28[k]*ecx2[8]} / nloc2; 0.0000;	
	/* printf("k=%d uloc1=%g u. */	uloc2=%g\n",k,uloc1,uloc2);	
	<pre>if(rholoc) { uloc = (rholoc1*uloc1 + jloc2 = rholoc2 * (uloc2</pre>	.1 + rholoc2*uloc2) / rholoc; oc2 - uloc);	
	uloc = 0.0000; /* printf("k=%d rholoc= */	rholoc=%g\n",k,rholoc);	
	loc1) oclbis =	<pre>(uloc1 * (1.000 - 0.500*delta_t/tau1) + uloc*0.500*delta_t/tau1)/nloc1;</pre>	
	oclbis = loc2) oc2bis =	0.000; (uloc2 * (1.000 - 0.500*delta_t/tau2) + uloc*0.500*delta_t/tau2)/nloc2;	
	else uloc2bis = 0.000; ulocbis = (rholoc1*uloc1* rholoc2*uloc2* rholoc1*uloc*0	= 0.000; (rholoc1*uloc1*(1.000 - 0.500*delta_t/taul) + rholoc2*uloc2*(1.000 - 0.500*delta_t/tau2) + rholoc1*uloc*0.500*delta_t/tau1 +	

dif9draw.c

Printed by sofonea from flumag3.mec.utt.ro	0
from flumag3	III.r
from flumag3	ည္ပ
from flum	Ĕ.
Printed by sofonea from flum	agg
Printed by sofonea from fl	Ę
Printed by sofonea fror	T E
Printed by sofonea	<u>5</u>
Printed by sofon	ea
Printed by so	õ
Printed by	SO
Printed	Ď
Prin	ted
	'n.

dif9draw.c

Apr 12 1999 18:42	dif9draw.c	Page 5	Apr 12 1999 18:42
./* rholoc2*uloc*0.	rholoc2*uloc*0.500*delta_t/tau2) / rholoc;		void test_ntot (void)
<pre>ntot += (nloc1 + nloc2); thotot += (tholoc1 + rholoc1 + rholoc1; tho1 += rholoc1; encequal += nongalloc; omegal += omegalloc; omegal += omegalloc; omegal += omegalloc;</pre>	c2) <i>}</i>		FILE *ptot; int k; double ncot=0.0000; char ntot_name[128]; sprintf(ntot_name, "P ptot = fopen(ntot_name,") for(k=0; k <nnodes_al< td=""></nnodes_al<>
omeganz += omeganzioc; 12 += 11oc2; u1 += u1oc2; u2 += u1oc2; u += u1oc; /*			ntot += fl0[k]+f fprintf(ptot,"%d %2 fclose(ptot);
u2bis += uloc2bis; ubis += ulocbis; */			void quiver (void)
ntot /= ((double) nnodes_y); rhotot /= ((double) nnodes_y); rho1 /= ((double) nnodes_y); rho2 /= ((double) nnodes_y); omegal /= ((double) nnodes_y);			double nloc1, nloc2, double ux1, uy1, ux2 dut, uy1, ux2 dut 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
omegal /= ((double) infordes_y) cmegan /= ((double) nnodes_y) j2 /= ((double) nnodes_y) i1 /= ((double) nnodes_y); u1 /= ((double) nnodes_y); u2 /= ((double) nnodes_y);	Th. (I).		sprintf(ux_name, #PUX sprintf(uy_name, #PUX sprintf(cx_name, "PCX sprintf(cx_name, "PCX sprintf(nx_name, "PCX sprintf(nx_name, "PXX
<pre>u /= ((double) nnodes_y); u2bis /= ((double) nnodes_y); ubis /= ((double) nnodes_y); fprintf(pntot, "%g\n" ntot); fprintf(pntot, "%g\n" ntot); fprintf(prhotot, "%g\n" rhotot); fprintf(prhot, "%g\n" rhoto); fprintf(prhot, "%g\n" rhot); fprintf(pnegal, "%l\n" rhot); fprintf(pnegal, "%l\n" rhot);</pre>			pux = fopen (uy_name, pux = fopen (uy_name, pxc = fopen (uy_name, pyc = fopen (cy_name, pyc = fopen (cy_name, pxn = fopen (ny_name, for (1=0); j <nnodes_y,< td=""></nnodes_y,<>
<pre>fprint(pomeganl, "klf\n", omegal); fprint(fpomegal, "klf\n", omegal); fprint(fpomegal, "klf\n", omegal); fprint(fpol, "klf\n", 12); fprint(ful, "klf\n", ul); fprint(ful, "klf\n", ul); fprint(ful, "klf\n", ul); fprint(ful, "klf\n", ul);</pre>	n. 2); n. 2);	3.	k = j*nnodes_x; for(i=0; i <nnode nloc1 = f10 nloc2 = f20 nloc2 = f20</nnode
<pre>fprintf(pu2bis,"%lf\n",u2bis); fprintf(pubis,"%lf\n",ubis);</pre>			if(nloc1) { ux1 = ()
fclose (phtot); fclose (prhotot); fclose (prhot); fclose (prhoto); fclose (pomegal); fclose (pomegal); fclose (pomegal); fclose (pomegal);			uy1 = (
folose (pll); folose (pul); folose (pul); /*			if(nloc2) (ux2 = 0
fclose(pubis); fclose(pubis); */			ny2 =
^			

yoid test_ntot (void)
FILE *ptot; int k; double ntot=0.0000; char ict_name[128]; sprintf(ntot_name,"PNTOT*s",id_name); ptot = fopen(ntot_name,"aw"); for(k=0; kcnnodes_all; k++)
ntot += f10[k]+f11[k]+f12[k]+f13[k]+f14[k]+f15[k]+f16[k]+f17[k]+f18[k];
<pre>fpintf(ptot,"%d %25.20lf\n",iter,ntot); fclose(ptot); }</pre>
void quiver(void)
FILE *pux, *puy, *pxc, *pyc, *pxn, *pyn; double nloc1, nloc2; double uxl, uyl, ux2, uy2, ux, uy, xc, yc; int i,j,k,ipas=2,jpas=2, xn, yn; char ux name1281; uv name1281;
char cx_name[128], cy_name[128]; char nx_name[128], ny_name[128]; char nx_name[128], ny_name[128]; char nx_name[128]
<pre>Sprintf(uv_name, "PDX%s.%odd", id_name, iter); sprintf(uv_name, "PDX%s.%odd", id_name, iter); sprintf(cx_name, "PDX%s.%odd", id_name, iter);</pre>
<pre>Sprintf(nx_name,"PNX%s.%06d",id_name,iter); sprintf(nx_name,"PNX%s.%06d",id_name,iter); sprintf(ny_name,"PNX%s.%06d",id_name,iter);</pre>
<pre>pux = ropen(ux_name,"w"); puy = fopen(uy_name,"w"); pxc = fopen(cx_name,"w"); pvc = fopen(cx_name,"w");</pre>
5
<pre>{ k = j*nnodes_x; for(i=0; i<nnodes_x; i+="ipas)</pre"></nnodes_x;></pre>
Ð
nloc2 = f20(k)+f21(k)+f22(k)+f23(k)+f24(k)+f25(k)+f26(k)+ f17(k)+f18(k); if(nloc1)
<pre>{ ux1 = (f11 K *ecx[1]+f12[k]*ecx[2]+f13[k]*ecx[3]+ f14[k]*ecx[4]+f15[k]*ecx[5]+f16[k]*ecx[6]+</pre>
<pre>f17 k]*eex(7]+f18[k]*eex[8]) / nlocl; uy1 = (f11[k]*eey[1]+f12[k]*eey[2]+f13[k]*eey[3]+ f14[k]*eey[4]+f15[k]*eey[5]+f16[k]*eey[6]+ f17[k]*eey[7]+f18[k]*eey[8]) / nlocl;</pre>
} e)se
ux1 = 0.0000; $uy1 = 0.0000;$
if (nloc2)
ux2 = (£21[k]*ecx[1]+£22[k]*ecx[2]+£23[k]*ecx[3]+ $£24[k]*ecx[4]+£25[k]*ecx[5]+£26[k]*ecx[6]+$
<pre>127 [k,*ecx[//+£28 [k,*ecx 8] / nloc2; uy2 = [£21[k]*ecy[4]+£22[k]*ecy[3]+</pre>

rage 8			
4			
ای			
alledraw.c			
dila			
,			
2			
Apr 12 1999 18:42			
2 196			
Apr			

15

Apr 6 1999 13:38 dif9fourier.c	Page 1

* dif9fourier.c fft * * * * * * * * * * * * * * * * * *	
include <stdio.h> include <stdib.h> include <stdib.h> include <stdih.h></stdih.h></stdib.h></stdib.h></stdio.h>	
<pre>void four1(double data[2*nnodes_x], int nn, int isign)</pre>	
<pre>int i, istep, j, m, mmax, n; double tempi, tempi; double theta, wi, wpi, wpr, wr, wtemp; n = 2 * nn; j = 1; for(i=1; i<=n; 1+=2)</pre>	
if(j>i)	
<pre>tempr = data[j-1]; tempi = data[j]; data[j-1] = data[i-1]; data[j-1] = data[i]; data[i-1] = tempr; data[i-1] = tempr;</pre>	
m = n/2; while ($m>=2$) && (j>= m))	
1 = 1-m; n /= 2;	
}	
mmax = 2; while $(n > mmax)$	
<pre>istep = 2 * mmax; theta = 6.2831853017959 / (isign * mmax); wpr = -2.000 * sin(0.5*theta) * sin(0.5*theta); wpi = sin(theta); wr = 1.000; wi = 0.000; for(m=1; m<= mmax; m+=2)</pre>	
for(i=m; i<=n; i+=istep)	
<pre>j = i+rmmax; tempx = wr * data[j-1] - wi * data[j]; tempx = wr * data[j-1] + wi * data[j-1]; tempi = wr * data[i-1] - tempi; data[j-1] = data[i-1] - tempi; data[i-1] = data[i-1] + tempr; data[i-1] = data[i-1] + tempi;</pre>	
wtemp = wr; wr = wr*wpr - wi*wpi + wr; wi = wi*wpr + wtemp*wpi + wi;	
mmax = istep; }	
) world resift (Anith to data (unedes x]. int n. int ision)	

	dif9fourier.c	Page 2
-		
int i, il, i2, i3, i4, n2p3; double cl, c2, hli, hlr, h21, h22, h2z, wis, wi double there wi war are aream:	3; 21, h2r, wis, wrs; wr. wremn:	
theta = 3.141592653589793 cl = 0.5;	/ (((double) n) / 2.000);	
11 = 13 = 1) } { 		
four1(data, n/2, 1);		
else {		
c2 = 0.5; theta = -theta;		
<pre>} wpr = -2.000 * sin(0.500*theta) wpi = sin(theta);</pre>	.heta) * sin(0.500*theta);	
wr = 1.000 + wpr; $wi = wpi;$		
n2p3 = n + 3; for(i=2; i<= n/4; i++)		
11 = 2*1 - 1;		
12 = 11+1; 13 = n2p3 - 12; 14 = 13 + 1;		
H 11	+ data[i3-1]); - data[i4-1]);	
(data (data - h1r	[1z-1 + data[4-1]; 11-1 - data[3-1]; - wis*h2r - wis*h2i;	
- hit	+ wrs*h2; + wis*h2r; - wrs*h2r + wis*h2r; + wrs*h2i + wis*h2r;	
c - wi*w c + wtem	w.t.	
п == 1)		
hlr = data[0]; data[0] = hlr + data[1]; data[1] = hlr - data[1];	57	
} else		
hir = data[0]; data[0] = c1*(hir + d data[1] = c1 * (hir - four1(data,n/2,-1);	data[1]); - data[1]);	

diffmain.c # diffmain.c # include setdio.h> # finclude setdio.h> # south setdio.h # south setdio.homble noil; double noil; double noil; # void diff_anmestint); # void test_distribution_functions(double noil); double noil; double noil; # south setdio.homble noil; double noil; double noil; # void intt_exty_aling_aquese_ext(void); # void intt_exty_aling_aquese_ext(void); # void intt_exty_aling_aquese_ext(void); # void compute_local_speeds(double nfil0]; double nfil0; double nfil0; # double drimmy; # for (k=0; kcmnodes_all; k++) # cor(k=0; kcmnodes_all; k++) # aloci(k = nfil0(k +nfilo(k +nfil

Abouble n2[], Souble n5[], Souble n8[], [], double double n8[], le nf18[], le nf18[], le nf28[], le nf28[], le nf28[],
double n2[], Souble n5[], Souble n5[], [[], double n8[]] double nf12[], ble nf12[], ble nf22[], ble nf25[], ble nf25[], ble nf25[], ble nf25[], ble nf28[]]
double n2[], double n8[], double n8[], f[], double double nf[], le nf[2[], le nf[3[], le nf[3], le nf[3[], le nf[3], le nf[3[], le nf[3], le nf[3
double n2[], double n5[], double n8[]) [[], double double n8[] he nf15[], he nf15[], he nf22[], he nf22[], he nf28[], he nf28[]]
<pre>compute_local_speeds(double nf10[], double nf11[], double nf12</pre>
<pre>int k; double dummy; for(k=0; k<nnodes_all; k++)<="" td=""></nnodes_all;></pre>
<pre>for(k=0; k<nnodes_all; k++)<="" td=""></nnodes_all;></pre>
uxloc1[k] = 0.0000000; uyloc1[k] = 0.000000; uxloc2[k] = 0.0000000; uyloc2[k] = 0.0000000; uxloc[k] = 0.0000000; uyloc[k] = 0.0000000;
else { switch(boundary_mode[k])
{

Apr 19 1999 09:20 dif9main.c	Page 2
nf14[k]*ecy1[4]+nf15[k]*ecy1[5]+nf16[k]*ecy1[6]+ nf17[k]*ecy1[7]+nf18[k]*ecy1[8]);	
<pre>uxloc2[k] = (nf21[k]*ecx2[1]+nf22[k]*ecx2[2]+nf23[k]*ecx2[6]+</pre>	
<pre>dummy = (mass1*nloc1[k])/tau1 + (mass2*nloc2[k])/tau2; if (dummy)</pre>	
<pre>tuxloc[k] = ((mass1 * uxloc1[k]) / tau1 +</pre>	
else { uxloc[k] = 0.0000; uyloc[k] = 0.0000;	
) if [nloc1[k])	
<pre></pre>	
<pre> uxloc1[k] = 0.0000; uyloc1[k] = 0.0000;</pre>	
<pre>if Inloc2[k])</pre>	
uxloc2[k] /= nloc2[k]; uyloc2[k] /= nloc2[k]; else	
uxloc2[k] = 0.0000; uyloc2[k] = 0.0000;	
/*; printf("k===%d %lf %lf %lf %lf %lf %lf\n", k,uxlooz[k],uylooz[k], uxlooz[k],uylooz[k],	
break; default:	
uylocky = break; } }	
) void compute_equilibrium_distributions2(void)	
<pre>int k; double dummy, uu, uul, uu2, s2; double ntl, nt2, net1, net2;</pre>	

Apr 19 1999 09:20 dif9main.c	Page 4
<pre>if(nloc2[k]) (uxloc[k] = (alpha*mass2*nloc2[k]*uxloc2[k] +</pre>	
<pre>dummy = (ecx2[1]*uxloc[k] * (1.000 - uu2); dummy = (ecx2[1]*uxloc[k] * (1.000 + three * dummy + dummy - uu2); dummy = (ecx2[2]*uxloc[k] * (1.000 + three * dummy + dummy - uu2); dummy = (ecx2[2]*uxloc[k] * (1.000 + three * dummy + dummy - uu2); dummy = (ecx2[3]*uxloc[k] * (1.000 + three * dummy + dummy - uu2); neq22[k] = w1*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq23[k] = w1*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq24[k] = w1*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq24[k] = w1*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq25[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2); neq28[k] = w2*nloc2[k] * (1.000 + three * dummy + dummy - dummy + dummy - dummy + dummy - dummy + dummy - dummy + dummy + dummy - dummy - dummy + dummy - dummy - dummy - dummy + dummy - dummy -</pre>	·
/* nt2 +=nloc1[k]; nt2 +=nloc2[k]; nt2 +=nq10[k]+neq12[k]+neq13[k]+neq14[k]+neq15[k] +neq10[k]+neq17[k]+neq18[k]; +neq10[k]+neq21[k]+neq28[k]; net2+=neq20[k]+neq21[k]+neq28[k]; +neq26[k]+neq21[k]+neq28[k]; if(iter > 99) &	; uxloc[k]);

8

Apr 19 1999 09:20 dif9main.c Page	5
<pre>printf("k=#d uyloc1=#g uyloc2=#g uyloc=#gyn",k,uyloc1[k],uyloc2[k],uyloc[k]); printf("#g *g *g</pre>	
/* ' printf("\n==== iter=%d k=%d ntl=%g netl=%g nt2=%g net2=%g\n\n", iter,k,nt1,net1,nt2,net2); */	
compute_equilib k; ble dummy, uu, ble nt1, nt2, n	
s2 = sqrt((double) 2); /* nt1 = (double) 0; nt2 = (double) 0; net1 = (double) 0; net2 = (double) 0;	· · · · · · · · · · · · · · · · · · ·
<pre>printf("\n==== iter=*d k=%d ntl=%g netl=*g nt2=*g net2=*g\n\n", iter,k,ntl,net1,nt2,net2); */</pre>	
<pre>for(k=0; k<nnodes_all; *="" cspeed12;="" cspeed22;<="" k++)="" pre="" uu="" uu2="three_over_two" uul="three_over_two" {=""></nnodes_all;></pre>	
<pre>neq10[k] = w0*nloc1[k] * (1.000 - uul); dummy = (ecx1[1]*uxloc[k] + ecy1[1]*uyloc[k]) / cspeed12; neq11[k] = w1*nloc1[k] * (1.000 + three * dummy + nne_uover_two * dummy * dummy - uul); dummy = (ecx1[2]*uxloc[k] + ecy1[2]*uyloc[k]) / cspeed12;</pre>	
= w1*nloc1[k] * (1.000 + three * dummy nine_over_two * dummy (ecx1[3]*uxloc[k] + ecy1[3]*uyloc[k]) * = w1*nloc1[k] * (1.000 + three * dummy nine_over_two * dummy	
<pre>dummy = (ecx1[4]*uxloc[K] + ecy1[4]*uyloc[K]) / cspeed12; neq14[K] = w1*nloc1[K] * (1.000 + three * dummy * dummy - uul); dummy = (ecx1[5]*uxloc[K] + ecy1[5]*uyloc[K]) / cspeed12; neq15[K] = w2*nloc1[K] * (1.000 + three * dummy * dummy - uul); dummy = (ecx1[6]*uxloc[K] * + ecy1[6]*uyloc[K]) / cspeed12;</pre>	

damy = (eczt[1] *uzloc[R] * (1.00 + three + dumny + dumy + dumy	Apr 19 1999 09:20 dif9main.c Page 6
= w2*hloci[k] * (1.000 + three * dummy + dummy - uul);	
= w2*nloc1[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc1[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc1[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy + dummy + dummy + dummy + dummy	= w2*nloc1[k] * (1.000 + three * dummy + nine over two * dummy * dummy - uul)
ext[8]*uxloc[k] + evg1[8]*uyloc[k]) cpeed[2]; = w2*nloc1[k] * (1.000 + three * dummy + dummy - uul); = w4*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w4*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w4*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w1*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy + dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy - uul); = w2*nloc2[k] * (1.000 + three * dummy * dummy + dummy + dummy + dummy + dummy + dummy + dummy	= (ecx1[7]*uxloc[k] + ecy1[7]*uyloc[k]) / cspeed12; k] = w2*nloc1[k] * (1.000 + three * dummy +
= w2*hloclk; "(1.000 + fhree * dummy * dummy - uul); = w0*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w1*hloclk; "(1.000 + three * dummy * dummy - uul); = w2*hloclk; "(1.000 + three * dummy * dummy * dummy + uul); = w2*hloclk; "(1.000 + three * dummy	<pre>nine_over_two * dummy * dummy - uul) = (ecx1[8]*uxloc[k] + ecy1[8]*uyloc[k]) / cspeed12;</pre>
= w0*nloc2[k] * (1.000 - uu2); ecx2[1]*uxloc[k] + ecy2[1]*uyloc[k]) / cspeed22; = w1*nloc2[k] * (1.000 + three * dummy * dummy - uu2); ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22; = w1*nloc2[k] * (1.000 + three * dummy * dummy - uu2); ecx2[3]*uxloc[k] + ecy2[3]*uyloc[k]) / cspeed22; = w1*nloc2[k] * (1.000 + three * dummy * dummy - uu2); = w1*nloc2[k] * (1.000 + three * dummy * dummy - uu2); = w2*nloc2[k] * (1.000 + three * dummy * dummy * dummy + uu2); = w2*nloc2[k] * (1.000 + three * dummy *	<pre>.kj = w2*nloc1[k] * (1.000 + three * dummy +</pre>
	= w0*nloc2[k] * (1.000 -
Decx2[3 wukloc[k] + ecy2[3 wyloc[k]) Cspeed22; Cspeed22 C	<pre>(ecx2[1]*uxloc[k] + ecy2[1]*uyloc[k]) / cspeed22; = w1*nloc2[k] * (1.000 + three * dummy +</pre>
	nine_over_two * dummy * dummy - uu2) = (ecx2[2]*uxloc[k] + ecy2[2]*uyloc[b]) / cspeed22; k = w1*nloc2[k] * (1.000 + three * dummy + dummy + three * dummy + dummy + three * dummy +
	#1.11002[h] 1:000
	eczistanciki (100 + threadelik) cspeed22;
= W_TIDCT[K] TIDOU + CLIDE = Warmy + dummy - uu2; = w2*nlocz[K] Valou + Luce = warmy + dummy - uu2; = w2*nlocz[K] * (1.00 + three * dummy * dummy * dummy - uu2; = w2*nlocz[K] * (1.00 + three * dummy * dummy - uu2; = w2*nlocz[K] * (1.00 + three * dummy *	eczelia uniocki dumy dumy dumy dumy dumy dumy dumy dumy
= w2*nloc2[K] * (1.00 + three * dummy + nine_over_two * dummy + nine_over_two * dummy * dummy - uu2); = w2*nloc2[K] * (1.000 + three * dummy * dummy - uu2); = w2*nloc2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy * dummy - uu2); co2[K] * (1.000 + three * dummy + three * dummy + three * dummy *	<pre>- w2.110c2[k] * (1.000 + cnree * dummy * dummy -</pre>
	<pre>= w2*nloc2[k] * (1.000 + three * dummy +</pre>
<pre>k] +neq15[k] k] +neq25[k] kg neq2=&g\n", [k] +neq17[k] +neq18[k], [k] +neq27[k] +neq28[k]); k, uxloc1[k], uxloc2[k], uxloc[k]) k, uyloc1[k], uyloc2[k], uyloc[k]) f15[k], f25[k],</pre>	<pre>[ecx2[8]*uxloc[k] + ecy2[8]*uyloc[k]) / cspeed22; = w2*nloc2[k] * (1.000 + three * dummy + dummy - uu2)</pre>
<pre>t]+neq15[k] k]+neq25[k] tg neq2=\$g\n", [k]+neq27[k]+neq28[k]); k,uxloc1[k],uxloc2[k],uyloc[k]) k,uyloc1[k],uyloc2[k],uyloc[k]) f15[k], f25[k],</pre>	/* nt1 +=nloc [k]:
<pre>k1+neq25[k] kg neq2=&g\n", [k]+neq17[k]+neq18[k], [k]+neq27[k]+neq28[k]); k,uxloc1[k],uxloc2[k],uyloc[k]) k,uyloc1[k],uyloc2[k],uyloc[k]) f15[k], f25[k],</pre>	nt2 +=nloc2[k] net1+=neq10[k] +neq11[k]+neq12[k]+neq13[k]+neq14[k]+neq15[k]
<pre>%g neq2=%g\n", [k]+neq17[k]+neq28[k]), [k]+neq27[k]+neq28[k]); k,uxloc1[k],uxloc2[k],uyloc[k]); k,uyloc1[k],uyloc2[k],uyloc[k]); f15[k], f25[k],</pre>	ruequ, tr, tr, tr, tr, tr, tr, tr, tr, tr, tr
<pre>[k] +neq17[k] +neq18[k], [k] +neq27[k] +neq28[k]); k, uxloc1[k], uxloc2[k], uxloc[k]); k, uyloc1[k], uyloc2[k], uyloc[k]); f15[k], f25[k],</pre>	, –
k,uxloc1[k],uxloc2[k],uxloc[k]) k,uyloc1[k],uyloc2[k],uyloc[k]) filocality,uyloc2[k],uyloc[k]) filocality, filocality, filocality,	<pre>iter,k.nloc1[k],neq10[k]+neq11[k]+ neq12[k]+neq12[k]+neq14[k]+neq15[k]+neq16[k]+neq17[k]+neq18[k], nloc2[k],neq20[k]+neq2[k]+neq25[k]+neq25[k]+neq25[k]+neq25[k]+neq28[k]);</pre>
k, uxloc1[k], uxloc2[k], uxloc[k]) k, uyloc1[k], uyloc2[k], uyloc[k]) fil5[k], fil5[k],	**
f16(k), 171(k), f18(k)), print("10 k), 121(k), f18(k)), f20(k), f21(k), f22(k), f23(k), f24(k), f25(k), f26(k), f27(k), f28(k)), print("10 k), f11(k), f11	k, uxloc1 [k], uxloc2 [k], uxloc[k]) k, uyloc1 [k], uyloc2 [k], uyloc[k])
<pre>f26(K),f22(K),f28(K); print("%g %g %g %g %g %g %g)\n", ff10(k),ff11(k),ff12(k),ff13(k),ff14(k),ff15(k), ff16(k),ff17(k),ff18(k)); print("%g %g %g %g %g %g %g %g %g %g ff26(k),ff21(k),ff22(k),ff23(k),ff24(k),ff25(k), ff26(k),ff27(k),ff28(k)); print(("%g %g %g %g %g %g %g %g %g)\n",</pre>	fieks filt fil
ff20[k], ff20[k], ff26[k], ff26[k],	print(IZ6K),IZ7(K),IZ8(K); print(IR9)
6[k],	<pre>til6[k],til7[k],til8[k]); ("%g %g %g %g %g %g %g\n", ff20[k],ff21[k],ff22[k],ff23[k],ff24[k]</pre>
neg10 k; neg11 k; neg12 k; neg13 k; neg14 k; neg15 k;	<pre>fi20(k),fi2/(k),fi28(k)); printf(#ag %g %g.", ned10(k).ned12(k).ned2(k).ned3(k).ned15(k).</pre>

Apr 19 1999 09:20 dif9main.c	Page 7
<pre>neq16[k],neq17[k],neq18[k]); printf("%gq %g %g</pre>	
runge_kutta(int key_point, double mass, double tau, double cspeed, double miss, double ni[], double ni2[], double ni4[], double ni5[], double ni5[], double ni7[], double ni5[], double ni5[], double ni7[], double nif1[], double nif2[], double nif7[], double nif5[], double nif7[], double nif5[], double si7[], double si7[], double si2[], double si7[], double si2[], double si3[], double si4[], double si4[], double si5[], double neq0[], double neq1[], double neq2[], double neq0[], double neq1[], double neq3[], double neq0[], double neq5[], double neq3[], double neq0[], double neq5[], double neq5[],	3177
<pre>int i, j, k; double ctau, cgrad, cgradx, cgradx; double cgradx2, cgradx2, cgradx2; double dummy_force; double prodscal;</pre>	
<pre>if(key_point) { ctau = delta_t / tau; cgrad = cspeed * delta_t / delta_x; cgradx = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_y; cgrady = cspeed * delta_t / (2.000*delta_x); cgradxy = cspeed * delta_t / (2.000*delta_x); dummy_force = (delta_t * cspeed) / (kboltz * temp); else }</pre>	
<pre>ctau = delta_t / (2.000 * tau); cgrad = cspeed * delta_t / (2.000 * delta_x); cgradx = cspeed * delta_t / (2.000 * delta_y); cgrady = cspeed * delta_t / (2.000 * delta_y); cgrady=cspeed*delta_t / (2.000 * delta_y); cgradx2 = cspeed * delta_t / (4.000 * delta_x); cgradx2 = cspeed * delta_t / (4.000 * delta_x); cgradx2 = cspeed * delta_t / (4.000 * delta_y); cgradx2 = cspeed * delta_t / (4.000 * delta_y); cgradx2 = cspeed * delta_t / (4.000 * delta_y); cgradx2 = cspeed * delta_t / (4.000 * delta_x); cgradxy2-cspeed*delta_t / (4.000 * celta_t x* delta_y * delta_y)); }</pre>	
<pre>for(k=0; k<nnodes_all; *="" 0:="" <="" bulk="" case="" first="" k++)="" order="" pre="" scheme="" switch(boundary_mode[k])="" upwind=""></nnodes_all;></pre>	
<pre>nff0[k] = sf0[k] - (nf0[k]-neq0[k])*ctau; if(ter < 0) /* second order attempt */</pre>	

Printed by sofonea from flumag3.mec.utt.ro dif9main.c Page 8

Apr 19 1999 09:20

Page 9				
Apr 19 1999 09:20 dif9main.c	nffi[k] = sfi[k] - (nfi[k]-neq1[k])*ctau - cgradx*ecx[i]*(ful[k]-nf2[k])*ctau - fi2[k] = sf2[k] - (nf2[k]-neq2[k])*ctau - cgradx*ecy[2]*(nf2[k]-nf2[nv4[k]]); nf5[k] = sf2[k] - (nf3[k]-nf2[k])*ctau - cgradx*ecx[3]*(nf3[nv1[k]]-nf3[k])* nf4[k] = sf4[k] - (nf4[k]-nf4[k])*ctau - cgradx*ecx[4]*(nf4[nv2[k]]-nf4[k]); nf5[k] = sf5[k] - (nf6[k]-neq4[k])*ctau - cgradx*ecx[5]*(nf5[k]-neq5[k])*ctau - cgradx*ecx[6]*(nf6[nv1[k]]); nff6[k] = sf6[k] - (nf6[k]-neq6[k]); nff6[k] = sf6[k] - (nf6[k]-neq6[k]); nff6[k] = sf6[k] - (nf6[k]-neq6[k]); cgradx*ecx[6]*(nf6[nv1[k]]); nff8[k] = sf8[k] - (nf7[k]-neqf[k]); nff8[k] = sf8[k] - (nf6[k]-nf6[nv4[k]); nff8[k] = sf8[k] - (nf6[k]-nf6[nv4[k]); nff8[k] = sf8[k] - (nf6[k]-nf6[k]); cgradx*ecx[7]*(nf7[nv1[k]]-nf7[k]); nff8[k] = sf8[k] - (nf8[k]-nf9[k]);	/* // // // // // // // // // // // // //	k) = sf0[k] - (nf0[k] = sf1[k] - (nf1[k] - (nf1[k] - (nf2[k] = sf2[k] - (nf2[k] = sf2[k] - (nf2[k] = sf2[k] - (nf3[k] = sf3[k] = sf3[k] - (nf3[k] = sf3[k] = (nf3[k] = sf3[k] = sf3[k] = sf3[k] - (nf3[k] = sf3[k] = sf3[k] -	/* /* /* /* /* /* /* /* /* /*

Apr 19 1999 09:20 dif9main.c	Page 10
<pre>default: nff0[k] = neq0[k]; nff1[k] = neq1[k]; nff3[k] = neq2[k]; nff3[k] = neq5[k]; nff4[k] = neq4[k]; nff6[k] = neq6[k]; nff6[k] = neq6[k];</pre>	
void lglb_coll(double mass, double tau, double cspeed, double nf0[], double nf1[], double nf2[], double nf3[], double nf4[], double nf5[], double nf6[], double nf7[], double nf8[], double nff8[], double nff8[], double nff8[], double nff8[], double nff4[], double nff8[], double nff8[], double nff8[], double neq5[], double neq0[], double neq4[], double neq5[], double neq6[], double neq4[], double neq8[],	
<pre>int k; double ctau = delta_t / tau;</pre>	
for(k=0; k <nnodes_all; k++)<="" td=""><td></td></nnodes_all;>	
nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau; nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau; nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau; nff3[k] = nf3[k] - (nf3[k]-neq2[k])*ctau; nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau; nff5[k] = nf5[k] - (nf6[k]-neq5[k])*ctau; nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau; nff7[k] = nf6[k] - (nf6[k]-neq6[k])*ctau; nff8[k] = nf8[k] - (nf8[k]-neq6[k])*ctau;	
<pre>printf("k=%d coll\n",k); printf("hf0=%g neq0=%g nff0=%g\n",nf0[k],neq0[k],nff0[k]); printf("nf1=%g neq1=%g nff1=%g\n",nf1[k],neq1[k],nff1[k]); printf("nf2=%g neq1=%g nff2=%g\n",nf1[k],neq3[k],nff2[k]); printf("nf4=%g neq4-%g nff2=%g\n",nf3[k],neq3[k],nff3[k]); printf("nf4=%g neq4-%g nff4=%g\n",nf4[k],neq3[k],nff4[k]); printf("nf5=%g neq6-%g nff6=%g\n",nf5[k],neq6[k],nff6[k]); printf("nf6=%g neq6-%g nff6=%g\n",nf5[k],neq6[k],nff6[k]); printf("nf8=%g neq6-%g nff6=%g\n",nf8[k],neq6[k],nff6[k]); printf("nf8=%g neq8=%g nff8=%g\n",nf8[k],neq8[k],nff6[k]); */</pre>	
void lglb_prop(double mass, double tau, double cspeed, double nff0[], double nff1[], double nff2[], double nff3[], double nff4], double nff5[], double nff0[], double nff7[], double nff8[], double nff0[], double nff7[], double nff8[], double nf0[], double nff1[], double nf5[], double nf6[], double nf7[], double nf8[])	
<pre>int k; double cgrad = cspeed * delta_t / delta_x;</pre>	
<pre>for (k=0; k<nnodes_all; k++)="" pre="" {<=""></nnodes_all;></pre>	
nf0[k] = nff0[k];	

Printed by sofonea from flumag3.mec.utt.ro

AifOmoin A	
00.00 00.00 to take	Apr 18 1889 08:20

Page 12

Printed by sofonea from flumag3.mec.utt.ro

Ap	000000 70 4			
Page 11				
Apr 19 1999 09:20 dif9main.c	switch (boundary_mode [k]) case 10:	void convection_diffusion(double mass, double tau, double cspeed, double nf0[], double nf1[], double nf2[], double nf2[], double nf3[], double nf3[], double nf3[], double nf3[], double nf4[], double neq2[], double neq4[], double neq5[], double neq4[], double neq5[], double neq5[],	int k; double ctau, cgrad, cgradx, cgradxs2, cgradys2, cgradxy; double cgradx2, cgradx2, cgradxy2; double delta_x2, delta_y2; double dummy_force; double prodscal;	<pre>ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_y;</pre>

<pre>cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_x; cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_y; cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y); cgradxy = cspeed * delta_t / (2.000*delta_x); cgradx2 = cspeed * delta_t / (2.000*delta_x); cgrady2 = cspeed * delta_t / (2.000*delta_y); cgradxy2 = cspeed * delta_t / (2.000*delta_x);</pre>	delta_x2 = delta_x * delta_x; delta_y2 = delta_y * delta_y;	<pre>for(k=0; k<nnodes_all; k++)<="" th=""><th>{ case 0: /* bulk */ nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +</th><th><pre>dcoer * (nf0[nv3[k]] - 2.000*nf0[k] + nf0[nv1[k]]) / delta_x2 + (nf0[nv2[k]] - 2.000*nf0[k] + nf0[nv4[k]]) / delta_y2); nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau - cgradx * ccx[1] * (nf1[k] - nf1[nv3[k])) +</pre></th><th>*</th><th>*</th><th>>1</th><th>* *</th><th>dcoer (nf5[nv3[k]] - 2.000*nf5[k] + nf5[nv1[k]]) / delta_x2 + (nf5[nv2[k]] - 2.000*nf5[k] + nf5[nv4[k]]) / delta_y2); nff6[k] - nf6[k] - neq6[k] * nf6[k] - nf6[k] + (nf6[k] - nf6[k]) + cgrady * cgv[6] * (nf6[k] - nf6[kv4[k]]) +</th><th>7[k] = gradx * grady *</th><th>gradx * grady *</th><th>dcoer (nf8[nv3[k]] - 2.000*nf8[k] + nf8[nv1[k]]) / delta_x2 + (nf8[nv2[k]] - 2.000*nf8[k] + nf8[nv4[k]]) / delta_y2);</th><th>/* / printf("k==%d %g printf("k==%d %g %g %g %g %g %g %g\n",k,</th></nnodes_all;></pre>	{ case 0: /* bulk */ nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +	<pre>dcoer * (nf0[nv3[k]] - 2.000*nf0[k] + nf0[nv1[k]]) / delta_x2 + (nf0[nv2[k]] - 2.000*nf0[k] + nf0[nv4[k]]) / delta_y2); nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau - cgradx * ccx[1] * (nf1[k] - nf1[nv3[k])) +</pre>	*	*	>1	* *	dcoer (nf5[nv3[k]] - 2.000*nf5[k] + nf5[nv1[k]]) / delta_x2 + (nf5[nv2[k]] - 2.000*nf5[k] + nf5[nv4[k]]) / delta_y2); nff6[k] - nf6[k] - neq6[k] * nf6[k] - nf6[k] + (nf6[k] - nf6[k]) + cgrady * cgv[6] * (nf6[k] - nf6[kv4[k]]) +	7[k] = gradx * grady *	gradx * grady *	dcoer (nf8[nv3[k]] - 2.000*nf8[k] + nf8[nv1[k]]) / delta_x2 + (nf8[nv2[k]] - 2.000*nf8[k] + nf8[nv4[k]]) / delta_y2);	/* / printf("k==%d %g printf("k==%d %g %g %g %g %g %g %g\n",k,
---	--	---	---	--	---	---	----	-----	--	------------------------------	-----------------	--	---

umag3.mec.utt.ro	Page 14
Printed by sofonea from flumag3.mec.utt.ro	dif9main.c

Apr 19 1999 09:20 dif9main.c Page 13	Apr 19 1999 09:20
k,nf0[k],nf1[k],nf2[k],nf3[k],nf4[k],nf5[k], nf6[k],nf7[k],nf8[k]); printf("k==\$d nff %g	kpezős - cgradk -
<pre>break; case 3: /* left wall */</pre>	**
nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +	Case 4: /*
	dcoef * (dcoef *)
⊳	- cgradx dcoef * (
<pre>dcoef * ((2.000*nf2[k] - 5.000*nf2[nv1[k]] +</pre>	- cgrady dcoef * (dcoef * friedy) - n:
<pre>dcoef * (</pre>	- ogracx dcoef * (dcoef * n
<pre>dcoef * (dcoef * (</pre>	- cgrady dcoef * (nff5[k] = n - cgradx
<pre>dcoef * (</pre>	- cgrady dcoef * (nff6[k] = n
<pre>dcoef * (</pre>	- cgrady dcoef * (
(2.000*n£7[k] - 5.000*n£7[nv1[k]] + 4.000*n£7[nv1[nv1[k]] - 1.07[nv1[nv1[k]]]) / delta_x2 + (n£7[nv2[k]] - 2.000*n£7[k] + n£7[nv4[k]]) / delta_y2); n££8[k] = n£8[k] - (n£8[k]-neq8[k])*ctau	* 3.0000

- cgradx * ecx[8] * (nf8[k] - nf8[k]) + - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) + dcoef * (2.000*nf8[k] - 5.000*nf8[nv1[k]] + 4.000*nf8[nv1[nv1[k]] -
5
<pre>case 4: /* right wall */ nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau +</pre>
, "pg.,
<pre>ccce1 (2.000*nf2[K] - 5.000*nf2[nv3[k]] +</pre>
ro ro
70
തത
(2.000*nf6[k] - 5. 4.000*nf6[ny3[ny3]ny3[ny3[ny3]ny3[ny3[ny3]ny3[ny3[ny3[ny3[ny3[ny3[ny3[ny3[ny3[ny3[

ag3.mec.utt.ro	Page 16
Printed by sofonea from flumag3.mec.utt.ro	dif9main.c

Apr 19 1999 09:20 ditymain.c Page 15	Apr 19 1999 09:20
<pre>nff8[k] = nf8[k] - (nf8[k]-neq8[k])*ctau - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]]) - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]) +</pre>	printf("K== neq0
<pre>dcoef * ((2.000*nf8[k] - 5.000*nf8[nv3[k]] + 4.000*nf8[nv3[nv3[k]]] - nf8[nv3[nv3[nv3[k]]]) / delta_x2 + nf8[nv3[nv3[nv3[k]]]) / delta_x2 + nf8[nv3[nv3[nv3[k]]]) / delta_x2 + nf8[nv3[v] - 2.00*nf8[k] + nf8[nv4[k]] / delta_v2);</pre>	printf ("K" printf ("K" k, nf
break; (inclination) = 1000 inclination break; (inclination)	*/ break;
} word first unwind (double mass, double tau, double cspeed,	case 3: /*
double nf0[], double nf1[], double nf2[], double nf3[], double nf4[], double nf5[], double nf6[], double nf6[], double nf6[], double nff0[], double nff0[], double nff0[],	nff0[k] = 1 nff1[k] = 1 - cgradx nff2[k] = 1
double nff5[], double nff5[], double nff6[], double nff7[], double nff8[], double neq0[], double neq1[], double neq2[], double neq8[], double neq4[], double neq5[], double neq6[], double neq4[], double neq6[]],	nff3(k)
int k; double ctau, cgrad, cgradx, cgradxs2, cgradys2, cgradxy; double cgradx2, cgrady2, cgradxy2; cgradxy	
<pre>ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_t / cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_x; cgradxs2 = cspeed * sqrt((double) 2) * delta_t / delta_y; cgradxy2 = cspeed * delta_t / sqrt(delta_x*delta_x * delta_y; cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x * delta_y * delta_y * delta_t / delta_t x * delta_t / delta_t x * delta_t / delta_t x * delta_t</pre>	
cyrady2 = cspeed * delta_t / (2.000*delta_y); cyrady2 = cspeed * delta_t / (2.000*delta_y); (2.000 * sqrt(delta_x*delta_x + delta_y*delta_y));	case 4: /* nff0[k] = nff1[k] =
<pre>for (k=0; k<nnodes_all; k++)<="" td=""><td></td></nnodes_all;></pre>	
<pre>case 0: /* bulk */</pre>	- cgradx nff4[k] = - cgrady nff5[k] =
<pre>nff2[k] = nf2[k] - (nf2[k] - nf1[nv3[k]]); nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]); - cgrady * ecy[2] * (nf2[k] - nf2[nv4[k]]);</pre>	- cgradx - cgrady nff6[k] = - cgrady
nis(x) = nis(x) - (nis(x) - neg(x)); - cgradx * ecx[3] * (nfg[nv1[k]] - nfg[k]); nff4[k] = nf4[k] - (nf4[k] - neg4[k]) * ccqrady * ecy[4] * (nf4[nv2[k]] - nf4[k]);	_ cgrady nff7[k] = _ cgradx
<pre>nff5[k] = nf5[k] - (nf5[k]-neq5[k])*ctau - cgradx * ecx[5] * (nf5[k] - nf5[nv3[k]]) - cgrady * ecy[5] * (nf5[k] - nf5[nv4[k]]); nff6[k] = nf6[k] - (nf6[k]-neq6[k])*ctau</pre>	Cgrady nff8[k] = - Cgrady - cgrady
- cgradx * ecx[6] * (nf6[nv1[k]] - nf6[k]) - cgrady * ecx[6] * (nf6[nv1[k]]); nff7[k] = nf7[k] - (nf7[k]) * ctan - cgradx * ecx[7] * (nf7[nv1[k]) - nf7[k])	break;
- cgrady * ecy[7] * (nf7[nv2[k]] - nf7[k]); nff8[k] = nf8[k] - (nf8[k] - neq8[k])*crau - cgradx * ecx[8] * (nf8[k] - nf8[nv3[k]]) - cgrady * ecy[8] * (nf8[nv2[k]] - nf8[k]);	void second_upwind(c

<pre>printf("k==\$d</pre>		<pre>case 4: /* right wall */ nff(k) = nf(k) - (nf(k) - neq(k))*ctau; nff(k) = nf(k) - (nf(k) - nf(k))*ctau - cgradx * ecx[1] * (nf(k) - nf(k))*ctau - cgradx * ecx[2] * (nf2[k] - neq(k)]); nff3[k] = nf3[k] - (nf2[k] - nf2[nv4[k]]); nff3[k] = nf3[k] - (nf3[k] - neq3[k])*ctau - cgradx * ecx[3] * (nf3[k] - neq3[k]); nff4[k] = nf3[k] - (nf3[k] - neq4[k]); nff5[k] = nf5[k] - (nf4[k] - neq4[k]) * ctau - cgradx * ecx[5] * (nf5[k] - neq4[k]); nff5[k] = nf5[k] - (nf5[k] - nf5[k]); - cgrady * ecy[5] * (nf5[k] - nf6[k]); nff5[k] = nf5[k] - (nf5[k] - nf6[k]); nff5[k] = nf5[k] - (nf6[k] - nf6[k]); - cgrady * ecy[6] * (nf6[k] - nf6[k]); nff6[k] = nf6[k] - (nf6[k] - nf6[k]); - cgradx * ecx[6] * (nf6[k] - nf6[k]) - cgradx * ecx[7] * (nf7[k] - nf7[k]); nff8[k] = nf8[k] - (nf8[k] - nf7[k]); nff8[k] = nf8[k] - (nf8[k] - nf7[k]); - cgradx * ecx[7] * (nf8[k] - nf7[k]); nff8[k] = nf8[k] - (nf8[k] - nf8[k]); - cgradx * ecx[8] * (nf8[k] - nf8[k]); nfall = nf8[k] - (nf8[k] - nf8[k]); - cgrady * ecy[8] * (nf8[k] - nf8[k]); - cgradx * ecx[8] * (nf8[k] - nf8[k]); - cgrady * ecy[8] *</pre>) void second_upwind(double mass, double tau, double cspeed, double nf0[], double nf1[], double nf2[], double nf0[], double nf4[], double nf5[],
) void

Apr 19 1999 09:20 dif9main.c	Page 17
double nff0[], double nff1[], double nff2[], double nff3[], double nff7[], double nff8[], double nef0[], double nef7[], double nef2[], double neq3[], double neq4[], double neq5[], double neq5[], double neq4[], double neq5[],	
<pre>1nt k; double ctau, cgrad, cgrady, cgrady; double cgrady, cgrady2, cgradxy2; double dummy_force; double prodscal;</pre>	
<pre>ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_y; cgrady = cspeed * delta_t / delta_y; cgradx2 = cspeed * delta_t / sgrt(delta_x*delta_x + delta_y*delta_y); cgradx2 = cspeed * delta_t / (2.000*delta_x); cgradx2 = cspeed * delta_t / (2.000*delta_y); cgradx2 = cspeed * delta_t / (2.000 * sgrt(delta_x*delta_x + delta_y*delta_y));</pre>	
<pre>for(k=0; k<nnodes_all; k++)<="" td=""><td></td></nnodes_all;></pre>	
-	
<pre>void first_centered(double mass, double tau, double nf2[],</pre>	
<pre>int k; double ctau, cgrad, cgradx, cgrady, cgradxy; double cgradx2, cgrady2, cgradxy2; double qradx2, cgradxy2; double prodscal;</pre>	
<pre>ctau = delta_t / tau; cgradx = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_x; cgrady = cspeed * delta_t / delta_x*delta_x + delta_y*delta_y; cgradx2 = cspeed * delta_t / (2.000*delta_x); cgradx2 = cspeed * delta_t / (2.000*delta_x); cgradx2 = cspeed * delta_t / (2.000*delta_y); cgradx2 = cspeed * delta_t / (2.000 * delta_y*delta_y); }</pre>	
<pre>for (k=0; k<nnodes_all; k++)<="" td=""><td></td></nnodes_all;></pre>	
case 0; /* bulk */ nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau; nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau; - cgradx2 * eox[l] * (nf1[nv1[k]] - nf1[nv3[k]]); nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau - cgrady2 * eoy[2] * (nf2[nv2[k]] - nf2[nv4[k]]); .nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau	

Printed by sofonea from flumag3.mec.utt.ro

Apr 19 1999 09:20 dif9main.c	Page 18
- cgradx2 * ecx[3] * (nf3[nv1[k]] - nf3[nv3[k]]); nff4[k] = nf4[k] - (nf4[k]-neq4[k])*ctau - cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]); nf5[k] = nf5[k] - (nf5[k]-neq5[k]) - nf5[nv3[k]] - cgradx2 * ecx[5] * (nf5[nv2[k]] - nf5[nv4[k]]); nff6[k] = nf6[k] - (nf6[k]-neq6[k]) - nf5[nv4[k]]); - cgradx2 * ecx[6] * (nf6[nv2[k]] - nf6[nv3[k]]) - cgradx2 * ecx[6] * (nf6[nv2[k]] - nf6[nv3[k]]); nff7[k] = nf7[k] - (nf7[k]-neq7[k]) * nf6[nv3[k]]); - cgradx2 * ecx[7] * (nf7[nv2[k]] - nf6[nv4[k]]); nff8[k] = nf8[k] - (nf8[nv2[k]] - nf7[nv3[k]]); - cgradx2 * ecx[7] * (nf7[nv2[k]] - nf7[nv3[k]]); - cgradx2 * ecx[8] * (nf8[nv2[k]] - nf7[nv3[k]]); - cgradx2 * ecx[8] * (nf8[nv2[k]] - nf8[nv4[k]]); break;	
case 3: /* left wall */	
<pre>nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;</pre>	
case 4: /* right wall */	
<pre>nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau;</pre>	

n flumag3.mec.utt.ro	Page 20
Printed by sofonea from flumag3.mec.utt.ro	dif9main.c

Apr 19 1999 09:20 dif9main.c Page 19	Apr 19 1999 09:20
word law friedrichs (double mass, double tau, double cspeed,	nff1[k] =
nf0[], double nf1[], doub nf3[], double nf4[], doub	_ cgrada; nff2[k] = (
double nIf(), double nIf(); double nIf(); double nIf(); double nIf(); double nIf();	- cgrady2 nff3[k] = (
nff6 neg0	നെ
<pre>double neq3[], double neq4[], double neq5[], double neq6[], double neq7[], double neq8[])</pre>	nff4[k] =
int k; double ctan. carady. carady. caradxy;	nff5[k] = (nf5[k] = (nf5[k]
cgradx2, cgrady2, cgr dummy_force;	- cgradx2 - cgrady2 nff6[k] =
double prouscar, ctan = delta t / tau:	- (nf6[k]
<pre>cspeed * delta_t / delta_x; cspeed * delta_t / delta_y;</pre>	_ cgrady? nff7[k] = (
<pre>cgradxy = cspeed * delta_t / sqrt(delta_x*delta_x + delta_y*delta_y); cgradx2 = cspeed * delta_t / (2.000*delta_x); cgradx2 = cspeed * delta_t / (2.000*delta_x);</pre>	- (nr/lk) - cgradk - cgradk
2 = cspeed * delta_t , (2.000 * ;	nff8[k] = (nf8[k]
<pre>for(k=0; k<nnodes_all; k++)<="" pre=""></nnodes_all;></pre>	- cgrady
<pre>switch(boundary_mode[k])</pre>	*/ .7 @##*
case 0: /* bulk */	2
nff0[k] = (nf0[nv1[k]]+nf0[nv3[k]]+nf0[nv2[k]]+nf0[nv4[k]]) / 4.000	0
nfilk] = (nfilm)	
- cgradx2 * ecx[1] * (nf1[nv1[k]] - nf1[nv3[k]]); nff2[k] = (nf2[nv1[k]]+nf2[nv3[k]]+nf2[nv2[k]]+nf2[nv4[k]]) / 4.000	nff2[k] = - (nf2[k
- (nf2 k -neq2 k)*ctau - cgrady2 * eqy[2] * (nf2[nv2[k]] - nf2[nv4[k]]); - cgrady2 * eqy[3] * (nf2[nv2[k]] - nf2[nv4[k]]);	- cgrady, nff3{k} = - (nf3(k')
	- cgradx
nffq[k] = (nff[nvi[k]) (nr.c)	
- cgrady2 * ecy[4] * (nf4[nv2[k]] - nf4[nv4[k]]); nff5[k] = (nf5[nv1[k]]+nf5[nv2[k]]+nf5[nv3[k]]+nf5[nv4[k]]) / 4.000	nff5[k] = (nf5[k
- (nf5[k]-neq5[k])*ctau - cgradx * evc[s] * (nf5[nv3[k]] - nf5[nv3[k]]) - cgradx * evc[s] * (nf5[nv3[k]] - nf5[nv4[k]])	- cgrady, - cgrady, nff6[k] =
- cgrady2 - ecv[5] - (nl2)inve[k]] - nl2)inve[k]]; nff6[k] = (nf6[k]/[k]]+nf6[nv2[k]]+nf6[nv4[k]]) / 4.000 - 'nf6[k],ncaf(k)] *rran	~ "
(inchr)	
nff7[k] = (nf7[nv1[k]) + nf7[nv2[k]] + nf7[nv3[k]] + nf7[nv4[k]]) / 4.000 - $(nf7[k] - neq7[k]) * cf.au$	- (nf7[k
(2 * ecx[7] * (nf7[nv1[k]] - nf7[nv3[k]]) (2 * ecy[7] * (nf7[nv2[k]] - nf7[nv4[k]]);	
<pre>nff8[k] = (nf8[nv1[k]]+nf8[nv2[k]]+nf8[nv3[k]]+nf8[nv4[k]]) / 4.000 - (nf8[k]-neq8[k])*ctau</pre>	- (nielk - cgradx
- cgradx2 * ecx[8] * (nf8[nv1[k]] - nf8[nv3[k]]) - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]); break;	- cgrady break;
case 3: /* left wall */	
<pre>nfff(k] = (nf0[nv1[k]]+nf0[k]+nf0[nv2[k]]+nf0[nv4[k]]) / 4.000 - (nf0[k]-neq0[k])*ctau;</pre>	void lax_wendroff(do

nffl[k] = (nfl[nvl[k])+nfl[k]+nfl[nv2[k])+nfl[nv4[k]]) / 4.000
~
- (nf2[k]-neq2[k])*ctau - cgrady2 * ecy[2] * (nf2[nv2[k]) - nf2[nv4[k]]); nff3[k] = (nf3[nv1[k]]+nf3[k]+nf3[nv2[k]]+nf3]nv4[k]]) / 4.000
- (nf3 k1-neg3 k1)*ctam - cgradx2 * ccx[3] * (nf3 nv1 k]] - nf3 k1); nff4 k] = (nf4 nv1 k1]+nf4 k1+nf4 k1+nf4 nv2 k1]+nf4 k104 k1]) / 4.000
LC LC
- (nf5 k -neq5 k)*ctau - cgradx2 * ecx[5] * (nf5 nv1 k]] - nf5 k) - cgrady2 * ecx[5] * (nf5 nv2 k]] - nf5 nv4 k]]); nff6 k] - (nf6 nv1 k]]+nf6 nv2 k]]+nf6 k]+nf6 nv4 k]]) / 4.000
cgre
(nf7[] cgrady cgrady 8[k] =
- (nrolk)-rego(k))-crau - cgradk2 * ecx[8] * (nf8[nv2[k]] - nf8[nv4[k]]); - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]); break;
case 4: /* right wall */
nff0[k] = (nf0[k]+nf0[nv3[k]]+nf0[nv2[k]]+nf0[nv4[k]]) / 4.000
r-t
0.1
ന
- (nf3K]-neq3(k])*ctem - cgradx2 * ecx[3] * (nf3[k] - nf3[nv3[k]]); nff4[k] = (nf4[k]+nf4[nv3[k]]+nf4[nv2[k]]+nf4[nv4[k]]) / 4.000
L/1
- (nf5 k -neq5 k)*ctau - cgradx2 * ecxf5 * (nf5 k] - nf5 nv3 k]]) - cgrady2 * ecxf5 * (nf5 nv2 k]] - nf5 nv4 k]]); nff6 k] = (nf6 k]+nf6 nv2 k]]+nf6 nv3 k]]+nf6 nv4 k]]) / 4.000
- (nf6 k -neq6 k)*ctau - cgradx* ecxf6 * (nf6 k - nf6 nv3 k) - cgrady2 * ecxf6 * (nf6 nv2 k - nf6 nv4 k); nff7 k = (nf7 k +nf7 nv2 k +nf7 nv3 k +nf7 nv4 k) / 4.000
- (nf/[k]-neq7[k])*ctau - cgradx2 * eoxf] * (nf7[k] - nf7[nv3[k]]) - cgrady2 * eoxf] * (nf7[k] - nf7[nv4[k]]); - cgrady2 * ecyf] * (nf7[nv2[k]] - nf7[nv4[k]]); nff8[k] = (nf8[k]+nf8[nv2[k]]+nf8[nv3[k]]+nf8[nv4[k]]) / 4.000
- (nf8[k]-neq8[k])*Ctau - cgradx2 * ecx[8] * (nf8[k] - nf8[nv3[k]]) - cgrady2 * ecy[8] * (nf8[nv2[k]] - nf8[nv4[k]]); break;
void lax_wendroff(double mass, double tau, double cspeed,

Apr 19	Apr 19 1999 09:20 dif9main.c.	Page 22
	+ xdunmy * cgradx * ecx[6] * (nf6[nv1[k]] - 2.000 * nf6[k] + nf6[nv3[k]]) - ydunmy * cgrady * ecy[6] * (nf6[nv2[k]] - 2.000 * nf6[k] + nf6[nv4[k]]); xdunmy = cgradx * ecx[7]; ydunmy = cgradx * ecx[7]; nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau - xdunmy * (nf7[nv1[k]] - nf7[nv3[k]]) + xdunmy * (nf7[nv1[k]] - nf7[nv3[k]]) + xdunmy * (nf7[nv2[k]] - nf7[k] + nf7[nv4[k]]) + ydunmy * cgradx * ecx[7] + ydunmy * cgradx * ecx[8]; ydunmy * cgradx * ecx[8]; + xdunmy * cgradx * ecx[8]; xdunmy = cgradx * ecx[8]; xdunmy = cgradx * ecx[8]; xdunmy = cgradx * ecx[8]; xdunmy = cgradx * ecx[8]; nff8[k] = nf8[k] - nf8[k] - nf8[k] + nf8[nv3[k]]) + xdunmy * (nf8[nv2[k]] - nf8[k] + nf8[nv3[k]]); + xdunmy * (nf8[nv2[k]] - nf8[k] + nf8[nv4[k]]); ydunmy * (nf8[nv2[k]] - nf8[k] + nf8[nv4[k]]); + xdunmy * cgradx * ecx[8]; + xdunmy * cgrady * ecy[8]; + xdunmy * cgrady * ecy[8];	
	<pre>nff0[k] = nf0[k] - (nf0[k]-neq0[k])*ctau; xdummy = cgradx2 * ecx[1]; pdummy = cgradx2 * ecx[1]; nff1[k] = nf1[k] - (nf1[k]-neq1[k])*ctau - xdummy = cgradx2 * ecx[1]; + xdummy = cgradx2 * ecx[1]; + (nf1[nv1[k]] - 2.000 * nf1[k] + nf1[k]); xdummy = cgradx2 * ecx[2]; nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau - ydummy = cgrady2 * ecy[2]; nff2[k] = nf2[k] - (nf2[k]-neq2[k])*ctau - ydummy = cgrady2 * ecy[3]; xdummy = cgrady2 * ecy[3]; nff2[k] = nf3[k] - (nf2[k]-neq2[k])*ctau - ydummy = cgrady2 * ecy[3]; xdummy = cgrady2 * ecy[3]; nff3[k] = nf3[k] - (nf3[k]-neq3[k])*ctau - xdummy * (nf3[nv1[k]] - nf3[k]); xdummy = cgradx2 * ecx[4]; ydummy = cgradx2 * ecx[4]; xdummy = cgradx2 * ecx[4]; ydummy = cgradx2 * ecx[4]; nff4[k] = nf4[k] - (nf4[k] - nf4[nv4[k]]); xdummy = cgradx2 * ecx[4]; ydummy * cgradx2 * ecx[5]; nff4[k] = nf4[k] - (nf4[k] - nf4[nv4[k]]); xdummy = cgradx2 * ecx[5]; nff5[k] = nf4[k] - (nf4[k] - nf5[k]); xdummy = cgradx2 * ecx[5]; nff5[k] = nf4[k] - (nf4[k] - nf5[k]); xdummy = cgradx2 * ecx[5]; nff5[k] = nf4[k] - (nf5[k] - neq5[k]); nff5[k] = nff5[k] - (nf5[k] - nf5[k]); nff5[k] = nff5[k] - (nf5[k] - nff5[k]); nff5[k] = nff5[k] - (nf5[k] - nff5[k]); nff5[k] = nff5[k] - (nff5[k] - nff5[k]); nff5[k] -</pre>	
	* cgrady 2[k]] - 2[k]] radx2 * e radx2 * e ff6[k] - (f ff6[k] - (f ff [nv * cgrady * cgrady	

27

Apr 19 1999 09:20 dlf9main.c	Page 23
* (nf6[nv2[k]] - 2.000 * nf6[k] + nf6[nv4[k]]); xdummy = cgrad42 * ecx[7]; ydummy = cgrad42 * ecx[7]; nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau - xdummy * (nf7[nv1[k]] - nf7[k]) + xdummy * (nf7[nv1[k]] - 2.000 * nf7[k] + nf7[k]) - ydummy * (nf7[nv2[k]] - 2.000 * nf7[k] + nf7[k]) + ydummy * (nf7[nv2[k]] - 2.000 * nf7[k] + nf7[nv4[k]]); xdummy = cgrad42 * ecx[8]; xdummy = cgrad42 * ecx[8]; ydummy = cgrad42 * ecx[8]; xdummy * (nf8[k] - (nf8[k] - nf8[k]) + nf8[k]) + xdummy * (nf8[nv1[k]] - nf8[k]) - ydummy * (nf8[nv1[k]] - nf8[k]) + ydummy * (nf8[nv1[k]] - nf8[k]) + ydummy * (nf8[nv2[k]] - nf8[nv4[k]]); break;	
case 4: /* right wall */	
yotnmny = cgrady2 * ecy[7]; $nff7[k] = nf7[k] - (nf7[k]-neq7[k])*ctau$	

Apr 19 1999 09:20	dif9main.c	Page 24
- xdummy * (nf + xdummy * cgrz * (nf7[k] - 2.(- ydummy * (nf + ydummy * cgrz * (nf7[nv2[k]] xdummy = cgrada2 ydummy = cgrada2 ydummy = cgrada2 ydummy = cgrada2 ydummy = cgrada2 + xdummy * (nff - xdummy * (nff - xdummy * (nff + ydummy * cgrz + ydummy * (grz + ydummy * (grz + ydummy * (grz + ydummy * (grz - ydum * (grz - ydummy * (grz - ydum * (gr	- xdummy * (nf7[k] - nf7[nv3[k]]) + xdummy * cgradx * ecx[7] * (nf7[k] - 2.000 * nf7[k] + nf7[nv3[k]]) - ydummy * (nf7[nv2[k]] - nf7[nv4[k]]) + ydummy * cgrady * ecy[7] * (nf7[nv2[k]] - 2.000 * nf7[k] + nf7[nv4[k]]); xdummy = cgrady2 * ecx[8]; ydummy = cgrady2 * ecy[8]; ydummy = cgrady2 * ecy[8]; nf8[k] = nf8[k] - (nf8[k] - nf8[k] + nf8[k] + nf8[k] - nf8[k] - nf8[k] - nf8[k] - nf8[k] + nf8[nv3[k]]) + xdummy * cgradx * ecx[8] * (nf8[k] - 2.000 * nf8[k] + nf8[nv3[k]]) - ydummy * cgrady * ecy[8] + ydummy * cgrady * ecy[8] * (nf8[k] - 2.000 * nf8[k] + nf8[nv4[k]]); } break;	
void leapfrog(double mass, double mf5] double mf5] double mf6] double mf6] double mf6] double mf76 double me46 double me46	mass, double tau, double cspeed, nf0[], double nf1[], double nf2[], nf3[], double nf4[], double nf5[], nf6[], double nf7[], double nf5[], nff8[], double nff1[], double nff2[], nff8[], double nff4[], double nff8[], nf90[], double nf7[], double nff8[], neq0[], double neq1[], double neq2[], neq8[], double neq4[], double neq8[], neq6[], double neq7[], double neq8[])	
int k; double ctau, cgrad, cgradx, cgrady, double ctau, cgrady, cgrady2; cgradx2; double dummy.force; double prodscal;	radx, cgrady, cgradxy; /2, cgradxy2;	
ctau = delta_t / tau; cgradx = cspeed * delta_t cgrady = cspeed * delta_t cgradxy = cspeed * delta_t cgradx2 = cspeed * delta_t cgradx2 = cspeed * delta_t cgradxy2 = cspeed * delta_t cgradxy2 = cspeed * delta_t	<pre>tau; * delta_t / delta_x; * delta_t / delta_y; * delta_t / delta_y; * delta_t / (2.000*delta_x); * delta_t / (2.000*delta_x); * delta_t / (2.000*delta_y); d * delta_t / (2.000 * sqrt (delta_x*delta_x + delta_y*delta_y));</pre>	
<pre>for(k=0; k<nnodes_all; k++)="" switch(boundary_mode[k])<="" td=""><td>1; k++) mode[k]) bulk */</td><td></td></nnodes_all;></pre>	1; k++) mode[k]) bulk */	
} void dif9 automaton(void)	Ŷ	
<pre>int autiter, key_point; autiter = 0;</pre>	מ	
while(autiter < nite: { test_distribution, test_distribution.	<pre>< niter_cycle) bution_functions(f10,f11,f12,f13,f14,f15,f16,f17,f18); bution_functions(f20,f21,f22,f23,f24,f25,f26,f26,f28);</pre>	
compute_local_spe	compute_local_speeds (f10, f11, f12, f13, f14, f15, f16, f17, f18, f20, f21, f22, f23, f24, f25, f25, f25, f26, f27, f28); compute_equilibrium_distributions();	

Aifomain Consolidation	Page 25	Anr 19 1999 00
Apr 18 1888 08:20	- ago to	200
switch(key_scheme)		leap
case 10: case	, 81	brea case 6
first_upwind(mas2, tau2, cspeed2, first_upwind(mas2, tau2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, f20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff neq20, neq22, neq22, neq23, neq25, neq25, neq26,	ff28,	- For
break; case 1: second_upwind(mass1, tau1, cspeed1, second_upwind(mill, fils, f	ff18,	rung
neq10, neq13, neq15, neq15, neq15, neq15, neq15, neq16); neq17, neq18); second_upwind(mass2, ta2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, ff20, ff21, ff22, ff23, ff25, ff25, ff25, ff27, f neq20, neq21, neq22, neq23, neq25, neq26,	ff28,	brea case 7 lglb
<pre>break; case 2: first_centered(mass1, tau1, cspeed1, first_centered(mass1, fil2, fil3, fil4, fil5, fil5, fil6, fil7, fil8, ffl0, fil1, fil2, fil3, fil5, fil5, fil6, fil7, fil fmil, fil1, fil2, new13, new14, new15, new16.</pre>	ff18,	1916
	ff28,	1916
. 0	ff18,	brea case conv
neq1, neq18, cspeed2, iedrichs (mass2, tau2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, f20, ff21, ff22, ff33, ff24, ff26, ff27, neq20, neq21, neq22, neq23, neq24, neq25, neq26,	ff28,	conr
break; case 4: lax_wendroff(mass1, tau1, cspeed1, 4, f15, f16, f17, f18, f10, f11, f12, f13, f14, f15, f16, f11, f13, f11, f11, f11, f11, f11, f11	ff18,	bree } test_dis test_dals
nedri, nedli, ospeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, f20, ff21, f22, f23, f24, f£5, f£66, f£7, neg20, neg21, neg22, neg23, neg24, neg25, neg26, neg27, neg25, neg25, neg25, neg25, neg25,	ff28,	compute compute switch (
break; case 5: leapfrog(mass1, tau1, cspeed1,		Case Case Case Case Case Case Case Case

Apr 19 1999 09:20 dif9main.c	Page 26
leapfrog(mass2, tau2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, neq20, neq20, neq22, neq23, neq24, neq25, neq26,	
<pre>break; case 6: store_distribution_functions(); be:</pre>	
runge_kutta(key_point, massl, taul, cspeedl, fl0, fl1, fl2, fl3, fl4, fl5, fl6, fl7, fl8, ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18, sf10, sf11, sf12, sf13, sf14, sf15, sf16, sf17, sf18, neq10, neq11, neq12, neq13, neq14, neq15, neq16,	
runge_kutta(key_point, mass2, tau2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, f20, f21, f22, f23, f24, f25, f126, f126, f27, f28, sf20, sf21, sf22, sf23, sf24, sf25, sf26, sf27, sf28, neq20, neq21, neq22, neq23, neq24, neq25, neq26,	
break;	
lglb_coll(mass2, tau2, cspeed2, f20, f21, f22, f24, f25, f26, f27, f28, f20, f21, f22, f23, ff24, ff26, ff27, ff28, neg20, neg22, neg22, neg24, neg25, neg25,	
£14, ££15, £15, £16,	
ď	
break; case 8:	
neq10, neq11, neq12, neq13, neq14, neq15, neq16, neq17, neq18; convection_diffusion_mass2, tau2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f28, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, neq20, neq21, neq23, neq23, neq25, neq26,	
break;	
<pre>test_distribution_functions(ff10, ff11, ff12, ff13, ff14, ff15, ff16,</pre>	
<pre>compute_local_speeds (ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18,</pre>	
case 0:	
<pre>case 10: first_upwind(mass1, tau1, cspeed1, first_upwind(mass1, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18, f10, f11, f12, f13, f14, f15, f17, f18, neq10, neq11, neq13, neq13, neq15, neq16, neq17, neq18);</pre>	

Apr 19 1999 09:20 dif9main.c	Page 27
<pre>first_upwind(mass2, tau2, cspeed2, first_upwind(mass2, ff21, ff23, ff24, ff25, ff26, ff27, ff28, f20, f21, f22, f23, f24, f25, f26, f27, f28, neq20, neq21, neq22, neq23, neq24, neq25, neq26, neq27, neq28);</pre>	
<pre>break;</pre>	
neq1/, neq1/, toqued2, second_upwind(mass2, tau2, cspeed2, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, f20, f21, f22, f23, f24, f25, f26, f27, f28, neq20, neq21, neq22, neq23, neq24, neq25, neq26,	
<pre>break; case 2: first_centered(mass1, tau1, cspeed1, first_filo, ffilo, ffilo, ffilo, ffilo, ffilo, ffilo, ffilo, filo, filo,</pre>	
first_centered(mass2, tau2, cspeed2, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff28, f20, f21, f23, f24, f25, f26, f27, f28, neq20, neq21, neq22, neq23, neq24, neq25, neq26,	
break; case 3: lax_friedrichs(mass1, tau1, cspeed1, lax_friedrichs(mass1, fil2, ff13, ff14, ff15, ff16, ff17, ff18, ff10, ff11, ff12, ff3, ff14, ff15, ff16, ff17, f18,	
neq10, neq11, neq12, neq13, neq14, neq15, neq15, neq15, neq15, neq15, neq15, neq15, neq17, neq18); lax_friedrichs(mass2, tau2, cspeed2, ff24, ff25, ff26, ff27, ff28, ff20, ff21, ff21, ff21, ff28, ff26, ff26, ff27, ff28, neq28, neq28, neq28, neq26,	
<pre>neq2/s, neq28); break; case 4: lax_wendroff(mass1, tau1, cspeed1, lax_wendroff(mass1, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18, f10, f11, f12, f13, f14, f15, f16, f17, f18,</pre>	
neq10, neq11, neq12, neq13, neq14, neq15, neq16, neq16, neq17, neq18; neq18; neq17, neq18; neq16, neq17, neq18; neq18; neq18; neq18; neq18; ff20, ff21, ff20, ff21, ff20, ff21, ff20, ff20, ff21, ff20, ff20, neq20, neq20, neq20, neq20, neq20, neq25, neq25, neq26,	
<pre>break; case 5; case 5; leapfrog(mass1, taul, cspeed1, leapfrog(mass1, fill, ff12, ff13, ff14, ff15, ff16, ff17, ff18, f10, ff11, ff12, ff3, ff14, ff15, ff16, ff17, ff18, f10, ff11, ff12, ff3, ff14, ff15, ff16, ff17, ff18,</pre>	
leapfrog(mass2, tau2, neq18); leapfrog(mass2, tau2, cspeed2, ff20, ff21, ff23, ff24, ff25, ff26, ff27, ff28, ff20, ff21, ff22, ff23, f24, f26, f27, f88, neq20, neq21, neq22, neq23, neq24, neq25, neq26,	
<pre>neq27, neq28); case 6: key_point = 1; runge_kutta(key_point, mass1, tau1, cspeed1,</pre>	

Apr 19 1999 09:20 dif9main.c	Page 28
ffl0, ffl1, ffl2, ffl3, ffl4, ffl5, ffl6, ffl6, ffl7, fl8, fl0, fl1, fl2, fl3, fl4, fl5, fl6, fl7, fl8, sl10, sl10, sl11, sl22, sl3, sl4, sl5, sl16, sl7, neq16, neq12, neq13, neq13, neq13, neq13, neq18, runge_kutta(key_point, mass2, tau2, cspeed2, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, fg0, f21, f22, f23, f24, f25, f26, f27, f28, sf20, sf21, sf21, sf24, sf25, sf26, sf27, neq26, neq12, neq22, neq23, neq24, neq25, neq26,	ff18, sf18, ff28, sf28,
break; case 7: break; case 7: break; case 8: convection_diffusion(mass1, tau1, cspeed1, ff15, ff16, ff17, ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff10, ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, neq16, neq17, neq18, neq12, neq14, neq15, neq16, neq16, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, ff20, ff21, ff22, ff23, ff24, ff25, ff26, ff27, neq20, neq21, neq22, neq22, neq22, neq25,	ff18, 5, ff28, 5,
ne < 6) ++; or lattice das like	
Rung [126 a stor key_ test test comp comp	
	7, ff18, 26, 26, 7, ff18, 7, ff28,

સ

Apr 19 1999 09:20 dif9main.c	Page 29
f10, f11, f12, f13, f14, f15, f16, f17, f18, ff10, ff11, ff18, s10, sf11, sf12, sf13, sf14, ff15, ff16, ff17, ff18, neq10, neq11, neq12, neq13, neq14, neq15, neq16,	
runge, kutta (key_point, mass2, tau2, cspeed2, f20, f21, f22, f23, f24, f25, f26, f27, f88, ff20, ff21, ff23, ff24, ff25, ff26, ff27, ff28, s20, s721, sf22, sf23, sf24, sf25, sf26, sf27, sf28, neq20, neq21, neq22, neq24, neq24, neq25, neq26,	
Key_point = 1; test_distribution_functions (ff10, ff11, ff12, ff13, ff14, ff15, ff16, ff17, ff18); test_distribution_functions (ff20, ff22, ff22, ff23, ff24, ff25, ff26,	
compute_local_speeds(fill),fill,fill,fill,fill,fill,fill,fill	
neq10, neq11, neq12, neq13, neq14, neq15, neq16, neq10, neq11, neq11, neq12, neq11, neq12, neq16, neq10, neq11, neq11, neq11, neq11, neq11, neq12, text, neq2, neq	
autiter++; iter++; */	
<pre>void main(void) { int isim, icycle;</pre>	
nnodes_x = a_nnodes_x[isim]; nnodes_y = a_nnodes_y[isim]; nnodes_y = a_nnodes_y[isim]; nmodes_all = a_nnodes_all[isim]; nmodes_all = a_nmodes_all[isim]; key_init = a_key_init[[isim]; key_force= a_key_force[[isim]; key_force= a_key_scheme[isim]; ncycles = a_ncycles[[isim]; niter_cycle = a_nter_cycle[[isim]; niter_init = a_nter_cycle[[isim]; niter_init = a_nter_cycle[[isim]; niter_init = a_nter_init[[isim]; length_x = a_length_x[isim]; length_y = a_length_x[isim]; delta_x = a_delta_y[isim]; delta_x = a_delta_y[isim]; force_x = a_force_x[isim]; force_x = a_force_x[isim]; massl = a_massl[isim]; massl = a_massl[isim]; massl = a_massl[isim];	
= l	

Apr	Apr 19 1999 09:20 dlf9main.c	Page 30
	<pre>cspeed2 = a_cspeed2(isim]; tau1 = a_tau1[isim]; tau2 = a_tau2[isim]; tau2 = a_tau2[isim]; nzerolleft = a_nzerolleft[isim]; nzerolleft = a_nzerolleft[isim]; nzerolight = a_nzerolleft[isim]; nzerolight = a_nzerolleft[isim];</pre>	
	<pre>cspeed12 = cspeed1 * cspeed1; cspeed22 = cspeed2 * cspeed2; cspeed1s = cspeed1 * sqrt((double) 2); cspeed2s = cspeed2 * sqrt((double) 2); cspeed2s2 = cspeed1s * cspeed1s; cspeed2s2 = cspeed2s * cspeed2s;</pre>	
	<pre>build_names(isim); init_lattice_functions(); init_ecx_nine(); init_arrays_nine_square(); init_arrays_nine_square_aux(); getavec_square();</pre>	
*	iter = 0; xv new(f10,f11,f12,f13,f14,f15,f16,f17,f18,xv_name,1.0001);	
*	test_ntot(); dif9_profile();	
	<pre>for(icycle=0; icycle<ncycles; icycle++)<="" td=""><td></td></ncycles;></pre>	
	/*	
	/* duiver();	
	<pre>*/ free_lattice_functions(); } free_nsim();</pre>	
~		

Printed by sofonea from flumag3.mec.utt.ro

Page 1	
Jul 17 1999 17:14 dif9.input	modes

Appendix C sw code swhead.h

Jul 9 1999 17:18

Jul 9 1999 17:18	rage i
*** ** *** *** *** *** *** *** *** *** *** *** *	
* * swhead.h * *	
* Definition of global variables *	
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\	
#define NNX 1024 #define NNY 192 #define NNODESI NNX * NNY + 1 #define NSIM 10	
#1fdef MAIN_HEADER	
char sim_name[40], asim_name[NSIM][40];	
<pre>char job_name[] = "sw.job", output_name[128], save_name[128], rez_name[128], profile_name[128], xv_name[128], wet_name[128], angle_name[128];</pre>	
double ecx[7], ecy[7], ecxx[7], ecxy[7], ecyx[7], ecyy[7];	
<pre>double f0[NNODES1], f1[NNODES1], f2[NNODES1], f3[NNODES1], f5[NNODES1], f6[NNODES1];</pre>	
<pre>double ff0[NNODES1], ff1[NNODES1], ff2[NNODES1], ff4[NNODES1], ff5[NNODES1], ff6[NNODES1];</pre>	
<pre>double floc[NNODES1], fgradx[NNODES1], fgrady[NNODES1], flap[NNODES1];</pre>	
<pre>double edip[NNODES1], egradx[NNODES1], egrady[NNODES1];</pre>	
int val[NNODES1];	
<pre>int nv0[NNODES1], nv2[NNODES1], nv3[NNODES1], nv4[NNODES1], nv5[NNODES1], nv6[NNODES1];</pre>	-
int boundary_mode[NNODES1];	
<pre>double rhomed, temp, temp_in, temp_fin, kappa, tau, tau3, ax, ay,</pre>	
<pre>int key_status, key_init, key_interaction, key_interaction_ini, key_interaction_fin, key_boundary, key_boundary_ini, key_boundary_fin;</pre>	
<pre>int akey_status[NSIM], aniter_init[NSIM], ansav[NSIM], ancycles [NSIM], aniter_cycle[NSIM], akey_interaction_ini[NSIM], akey_interaction_ini[NSIM], akey_interaction_fin[NSIM], akey_interaction_fin[NSIM],</pre>	
<pre>double arhomed[NSIM], atemp_in[NSIM], atemp_fin[NSIM], akappa[NSIM], atau[NSIM], aax[NSIM], amx[NSIM], amy[NSIM], arradmiulwalli[NSIM], agradmiulwalli[NSIM], agradmiulwalli[NSIM], agradmiulwallz[NSIM];</pre>	

<pre>double arl[101], aalphal[101], aalphad1[101], atanalphal[101], ar2[101], aalpha2[101], aalphad2[101], acosalpha1[101], acosalpha2[101];</pre>
int timecount;
int nsim, niter_init, nsav, ncycles, niter_cycle, iter, niter;
int key, scale;
const double b = 6.000000000 , bl = 7.000000000 , apsi = $9.0/49.0$, bpsi = $2.0/21.0$;
<pre>const int nnodx = NNX, nnody = NNY, nnodes = NNX * NNY, nnodes1 = NNX * NNY +1, nnodesxy1 = NNX * (NNY - 1), nnodes_aux = NNX * NNY + 2 * (NNY + NNX + 1) + 1;</pre>
const double dnnodx = (double) NNX, dnnody = (double) NNY;
const double three_over_two = 3.00 / 2.00, nine_over_two = 9.00 / 2.00;
∳else
extern char sim_name[40], asim_name[NSIM][40];
<pre>extern char job_name[],</pre>
<pre>extern double ecx[], ecx[], ecxx[], ecxx[], ecyx[], ecyy[];</pre>
extern double f0[], f1[], f2[], f3[], f4[], f5[], f6[];
extern double ff0[], ff1[], ff2[], ff3[], ff4[], ff5[], ff6[];
extern double floc[], fgradx[], fgrady[];
extern double edip[], egradx[], egrady[];
extern int val[];
extern int nv0[], nv1[], nv2[], nv3[], nv4[], nv5[], nv6[];
extern int boundary_mode[];
extern double rhomed, temp_in, temp_fin, kappa, tau, tau3, ax, ay, mm, mm, mx, my, tranhingall; cranhingall; cranhingall;
double gradminiwaili, gradminiwaili, yradminiwaili, yradminiswaili
<pre>extern int key_status, key_init, key_interaction_ini, key_interaction_fin, key_interaction, key_interaction_ini, key_interaction_fin, key_boundary, key_boundary_ini, key_boundary_fin;</pre>
<pre>extern int akey_status[], aniter_init[], ansav[], ancyclos[], aniter_cote[], akey_initer_action_ini[], akey_interaction_ini[], akey_interaction_ini[], akey_boundary_fin[], akey_boundary_ini[], akey_boundary_fin[];</pre>
extern double arhomed[], atemp_in[], atemp_fin[], akappa[], atemp_atem[], amv[], amv[], amv[], amv[], agradmiulwall1[], agradmiulwall1[], agradmiulwall2[], agradmiulwall2[], agradmiulwall2[],
extern int av[], as[], achi[], atime[];
<pre>extern double ar1[], aalpha1[], aalphad1[], atanalpha1[], ar2[], aalpha2[], atanalpha2[], acosalpha1[], acosalpha2[];</pre>

extern int naim, niter_init, neav, noycles, niter_cycle, iter, niter; extern int key, scale; extern const double b, b1, apsi, bpsi; extern const din nock, nnock, nnock, nnockes, nnock	swhead.h
extern const double b, bl, apsi, bpsi; extern const int nnode, nnodes, nnodesi, nnodesxyl, extern const dnnodx, dnnody; axtern const double three_over_two, nine_over_two; ndif	les, nite
OVET_LWO;	odes1,
	ver_tw

Jul 13 1999 09:19 swinout.c	Page 1		'
/*************************************		<u> </u>	
<pre>#include <stdio.h> #include <stdiib.h> #include <string.h> #include "swhead.h" void sw input (void)</string.h></stdiib.h></stdio.h></pre>			
<pre>fint i, j; FILE *fin; fin = fopen(job_name, "r"); fscanf(fin, "\$d\n", fnsim); if(nsim < 1) exit(1); fx(1); fx(1); fx(1); fx(1); fx(1);</pre>			•
fscanf (fin, "\$d \$s\n", &key_status, sim_name); fscanf (fin, \$d\$ \$d \$d \$d\n", \$niter_init, \$nsav, \$ncycles, \$niter_cycle); fscanf (fin, \$d\$ \$d \$d \$d\n", \$niter_init, \$nsav, \$ncycles, \$niter_cycle); fscanf (fin, \$d\$ \$d \$d \$d\n", \$key_init, \$nsav, \$nc, \$nc, \$nc, \$nc, \$nc, \$nc, \$nc, \$nc			
<pre>while (sim_name[j]); aniter_init[i] = niter_init; ansav[i] = nsav; ansav[i] = nsav; ansav[i] = nsav; anter_cycles[i] = niter_cycle; akey_init[i] = key_init; akey_init[i] = key_init; akey_init=acation_ini[i] = key_interaction_ini; akey_interaction_ini[i] = key_interaction_ini; akey_interaction_ini[i] = key_interaction_ini; akey_boundary_ini[i] = key_boundary_ini; akey_boundary_ini[i] = key_boundary_ini; atemp_in[i] = temp_in; atemp_in[i] = temp_in; atal[i] = temp_in; axal[i] = ka; axi[i] = ax;</pre>	÷		
_			
void sw_output (void) { int i; FILE *fout;			
		,	

Jul 13 1999 09:19 Page 2
<pre>fout = fopen("sw.out","w"); fprint(fout,"*d\n",nsim); for(i=0; i<nsim; i++)<="" pre=""></nsim;></pre>
<pre>fprintf(fout, "%d %s\n", akey_status[i], asim_name[i]); fprintf(fout, "%d %d % %d\n" aniter_init[i], ansav[i],</pre>
void build_names(int isim)
<pre>int irhomed.itemp_in.itemp_fin.itau.ikappa.iax.iay.imx.imy; int igradmiulwall1.igradmiulwall1.igradmiulwall2;</pre>
<pre>irhomed = (int) (arhomed[isim] * 100.00); ithomed = (int) (atemp_fin[isim] * 1000.00); itemp_fin = (int) (atemp_fin[isim] * 1000.00); itemp_fin = (int) (atemp_fin[isim] * 1000.00); itau = (int) (atemp_fin[isim] * 1000.00); ix = (int) (axapatisim]*1000.00); ix = (int) (axapatisim]*1000.00); ix = (int) (axapatisim]*1000.00); imx = (int) (axapatisim]*1000.00); imx = (int) (axapatisim]*1000.00); imx = (int) (axapatisim]*1000.00); ix = (int) (axapatisim]*100000; ix = (int) (axapatisim]*10000; ix = (int) (axapatisim]*100000; ix = (int) (axapatisim]*10000</pre>
int read_status(void)
<pre>fILE *fsav; if((fsav = fopen(save_name, "r")) == NULL); </pre>
return 1; /* if(
<pre>(fread(&iter, sizeof iter, 1, fsav) != 1) (fread(f0, sizeof f0, 1, fsav) != 1) (fread(f1, sizeof f1, 1, fsav) != 1) (fread(f2, sizeof f2, 1, fsav) != 1) (fread(f3, sizeof f3, 1, fsav) != 1) (fread(f3, sizeof f3, 1, fsav) != 1) </pre>

ო

				 	<u>.</u>
Page 4	7.age 4				
	:				
swinout.c	SWINOUT.C				
1111319990919	Jul 13 1999 09:19				

4

swinit.c

Jul 9 1999 17:18

Page 2

<pre>#include <stdio.h> #include <stdlib.h> #include <math.h> #include <math.h> #include <string.h></string.h></math.h></math.h></stdlib.h></stdio.h></pre>	
<pre>#include "swhead.h" void print_tavec(int kk, int nv[])</pre>	
<pre>{ int i, j, k; FILE *fout; fout = fopen(rez_name, "aw"); fprintf(fout, "\nnv%id\n", kk); k=0; for('=1; i<=nnody; i++)</pre>	
for (i=1;i<=nnodx;i++) {	
<pre>k++; fprintf(fout,"%3d ",nv[(j-1)*nnodx+i]); fprintf(fout,"\n");</pre>	
fclose (fout);	
void getavec_hex(void)	
int i, j, k, il; int nn[nnodes+1+2*(nnody+nnodx+1)+1+nnodes]; int in = nnodx+1, n0, n1, n2, n3, n4, n5, n6, nn1, for(i=1;i<=nnody;i++)	ip;
<pre>' in++; nn[in]=i*nnodx; for(j=1;j<=nnodx;j++)</pre>	
in++; nn[in]=(i-1)*nnodx+j;	
in++; nn[in]=(1-1)*nnodx+1;	
in=0; for(i=1;i<=nnodx;i++)	
in++; nn[in]=(nnody-1)*nnodx+i;	
in++; nn[in]=nn[i]; in=(nnodx+1)+nnody*(nnodx+2); for(i=1;i<=nnodx+1;i++)	
in++; nn[in]=nn[nnodx+1+i];	
in+; nn(in)=1; n0=0;	
<pre>for(i=1;1<=nnody;1++)</pre>	
{ nnl=11+j; ip=1*2; no+; no+; nl=nnl+1;	

n2=nn1-(nnodx+2-ip); n3=n2-1; n4=nn1-1; n5=nn1+(nnodx+1+ip); n6=nn1+(nnodx+1+ip); nv0[n0]=nn(n1); nv1[n0]=nn[n1]; nv3[n0]=nn[n2]; nv3[n0]=nn[n2]; nv3[n0]=nn[n2]; nv4[n0]=nn[n3]; nv6[n0]=nn[n4];	
/*) print_tavec(0,nv0); print_tavec(1,nv1); print_tavec(3,nv2); print_tavec(4,nv3); print_tavec(4,nv4); print_tavec(5,nv5); print_tavec(6,nv6); */	
<pre>void init_arrays_hex() { FILE *fsav, *frez; int i, j, k, c=nnody, 2, jc=nnody/2, scala = 14, key_rho, layer_height; double rand_coef = 0.1, rho_high, rho_low; double x, y, raza, raza_max=nnody*sqrt(3.)/4.,xc,yc; iter = 0; frez=fopen(rez_name, "a"); frez=fopen(rez_name, "a"); freyintf(frez=fopen(rez_name, "a"); freyintf(frey="\n") = init_arrays\n");</pre>	
ev(1)=0.00; ev(2)=0.50; ev(3)=ev(2) ev(3)=ev(2); ev(4)=ev(2); ev(4)=0.00; ev(5)=ev(3); ev(5)=ev(3); ev(5)=ev(3); ev(5)=ev(3); ev(6)=ev(3); ev(6)=ev(3); ev(6)=ev(3);	
acxx[i] = ecx acyx[i] = ecx acyx[i] = ecy acyy[i] = ecy i=1; i <= 6;	
<pre>switch (key_init)</pre>	

swinit.c

Jul 9 1999 17:18

Page 4

Jul 9 1999 17:18 Swinit.c Pa	Page 3
(((double) rand() / (double) (double) (double) (double)	
<pre>break; case 1: /* init full square */ case 2: /* init void square */</pre>	
rho_high = 1.1 * rho_high; k=0; for (j=1; j<=nnody; j++) for (j=1; i<=nnody; j++)	
* (1.+rand_coef * (((double) rand() /(double) RAND_MAX) - 0.5))	
for (j=nnody/4-2; j<=(3*nnody)/4+2; j++) for(i=nnodx/4-2; i<=(3*nnodx)/4+2; i++)	
<pre>{ k=nnody*1+j; f0[k] = 1.000/(b1)* (1.+rand_coef*(((double) rand() /</pre>	
<pre>fox j=nnody/4; j<=(3*nnody)/4; j++) for(i=nnodx/4; i<=(3*nnodx)/4; i++)</pre>	
<pre>{ k=nnody*i+j; f0[k] = 1.000(b1)* (1.+rand_coef*((double) rand() /</pre>	
break; case 3: /* init drop */ case 4: /* init bubble */	
<pre>traza=((double) scala)*raza_max/16.; xo=((double) nnodx)/2.+((double) (nnody-(nnody/2)*2))/2.; yo=((double) nnody)*sqrt(3.)/4.;</pre>	
$K=0;$ for $(j=1; j \le nnody; j++)$	
<pre>y=sqrt(3.)*((double) j)/2.; for(i=1; i <= nnodx; i++)</pre>	
<pre>{ x=((double) i)+((double) (j-(j/2)*2))/2.; k++; if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=(raza+2.)*(raza+2.))</pre>	•
key_rho=2; else_rho=3; if((x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza) key_rho=1; switch (key_rho)	
<pre>case 1: f0[k] = rhomed * 1.35/(b1)*. (1.+rand_coef*(((double) rand() /</pre>	
<pre>break; case 2: case 3: case 4: case 3: case 4: case 5: case 5: case 5: case 5: case 5: case 5: case 6: case 6: case 6: case 6: case 6: case 6: case 7: case 6: case 6: case 7: case 7</pre>	
<pre>break; case 3: f0[k] = rhomed * 0.65/{ bl)*</pre>	

swinit.c

Jul 9 1999 17:18

		я.		
<pre>(1.+rand_coef*(((double) rand() / (double) RAND_MAX) - 0.5); for (j=layer_height; j<= layer_height+1; j++) for(i=1; i<=NNLUN; i++) for(i=1; i<=NNLUN; i++) for(i=1) i<=NNLUN; i++) for(i=1; i<=NN</pre>	<pre>for (j=layer_height+4; j<= NNLAT; j++)</pre>	<pre>fprintf(frez,"rho.high = %f rho_low = %f\n",rho_high,rho_low) raza=((double) scala)*raza_max/16.; raza=((double) snulN)*2.+((double) (NNLAT/2)*2))/2.; xc=((double) NNLUN)*2.+((double) (NNLAT/2)*2))/2.; k=0; for(j=1; j <= NNLAT; j++) for(j=1; j <= NNLAT; j++) for(j=1; j <= NNLUN; i++) for(j=1; i <= NNLUN; i++) for(j=1; i</pre>	<pre>k++; k++; k++; f(x-xc)*(x-xc)+(y-yc)*(y-yc)<=(raza+2.)*(raza+2.)) key_rho=2; key_rho=3; key_rho=3; key_rho=3; key_rho=3; key_rho=3; key_rho=3; key_rho=3; f(x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza) key_rho=3; f(x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza) f(x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza) f(x-xc)*(x-xc)+(y-yc)*(y-yc)<=raza*raza) f(x-xc)*(x-xc)+(y-yc)*(y-yc)+(y-</pre>	Dreak; £0[K] = rho_med/(bl)* {1.+rand_coef*((double) rand() /

Jul 9 1999 17:18	swinit.c	Page 8
for (j=2; j<=NNLAT-1; j++) if (j % 2)	(++	
{ boundary_mode[(j-1)*NNLUN+1] boundary_mode[j*NNLUN] = 6; } else	[1,*NNLUN+1] = 5; $[NLUN] = 6;$	
boundary_mode[(j-1)*nnodx+1] boundary_mode[j*nnodx] = 4;	1)*nnodx+1] = 3; nodx] = 4;	
boundary_mode[1] = 7; boundary_mode[nnodx] = 8; boundary_mode[nnodx] = 9; boundary_mode[nnodes] = 10;	8, 9; 10;	
*/ break;		
•		

swdraw.c

Jul 9 1999 16:41

Page 2

Jul 9 1999 16:41 Swdraw.c Page	-
#include <stdio.h> #include <math.h> #include "swhead.h"</math.h></stdio.h>	
<pre>void xv(double n0[], double n1[], double n2[], double n3[],</pre>	
<pre>fric *fxv; char xv_name[128]; int i,k,val; sprintf(xv_name, "%s.%05d", arg_name, iter); fxv = fopen(xv_name, "wt"); fxv = fopen(xv_name, "wt"); fprintf(fxv, "P2\n%3d%4d\n63\n", nnody); i=0;</pre>	
<pre>for(k=1; k<=nnodes; k++)</pre>	
if (val > 63) val = 63; if (val < 0);	
/* val = 0;	
i=0; fprintf(fxv,"\n"); }	
<pre>1f(1)</pre>	
<pre>/*sprintf(xv_name, "%s.%05dp", arg_name, iter); fxv = fopen(xv_name, "wt"); fprintf(fxv, "p2\n%3d%4d\n63\n", nnlun, nnlat); i=0;</pre>	
<pre>for(k=1; k<=nnod; k++) {</pre>	
<pre>1++; val = (floor((fl[k]*ecx[1]+f2[k]*ecx[2]+f3[k]*ecx[3]+</pre>	
<pre>printf("%25.16f\n", (f1[k]*ecx[1]+f2[k]*ecx[2]+f3[k]*ecx[3]+</pre>	
<pre>if(val > 63) val = 63; if(val < 0) val = 0; val = 0; val = 63 - val;</pre>	7
<pre>fprintf(fxv, "%3d", val); if(i == 15)</pre>	
1=0; fprintf(fxv,"\n"); }	
<pre>i f(1)</pre>	
<pre>}' void profile(double n0!), double n1[], double n2[], double n3[],</pre>	
]

double	double n4[], double n5[], double n6[], char arg_name[])	
<pre>fILE *fxv; char xv.name[128]; int i,k; double val; sprintf(xv.name, "% fxv = fopen(xv.name) for(i=1; i<=nnody;</pre>	<pre>fILE *fxv; int i, k; int i, k; double val; sprintf(xv_name, "%s %05d", arg_name, iter); fxv = fopen(xv_name, "wt"); for(i=1; i<=nnody; i++)</pre>	
k=(1-1)* val = n0 fprintf(k=(1-1)*nnodx+nnodx/2; val = n0[k]+n1[k]+n2[k]+n3[k]+n4[k]+n5[k]+n6[k]; fprintf(fxv,"\$3d %1f\n",1,val);	
fclose(fxv);		
void minko (void)	(g)	
FILE *fwet, *falpha; int pix(NNODES1); double *rholoc, s, v, double height, width, int in, imminus1, ihm int i, i, k; int i', is;	*falpha; BES1; or, S, V, coef, alphal, alpha2, a, b, c, fa, fb, fc; tt, width, galpha, gtan, gcos; tt, width, galpha, gtan, gcos; nusl, ihminus2, ihplus1, ihplus2, iw;	
timecount++; iv = 0;		
for (i=1; i<=	for(i=1; i<=nnodes; i++)	
rholoc =	<pre>rholoc = f0[i] + f1[i] + f2[i] + f3[i] + f4[i] + f5[i] + f6[i]; if(rholoc > 3.5)</pre>	
pix[i]	[[1] = 1; ++;	
else		
pix[1]	(1) = 0;	
,		
is = 0; for(i=1; i<=nnodes;	=nnodes; 1++)	
(if (pix[i])	11)	
) IT (if(!pix[nv1[i]])	
	18++;	_
11 (1	} (!pix[nv4[i]])	
	is++;	
swit	<pre>switch(boundary_mode[i])</pre>	
~ ŏ		
	if (lpix[nv3[i]])	
	1.5++;	
	if(!pix[nv5[i]])	_

\sim
3.mec.utt.rc
. •
_
=
\rightarrow
-
11
\mathbf{c}
as -
_
_
_
_
_
.,
3
v,
m.
flumag
_
=
_
_
•
from
_
=
0
_
_
-
ത
Ω.
ဓရ
ည်
nea
onea
lonea
fonea
ofonea
sotonea
sotonea
sotonea
y sofonea
y sof
by sofonea
y sof
y sof
y sof
y sof
ed by sof
ed by sof
ed by sof
ed by sof
ed by sof
rinted by sof
ed by sof
rinted by sof
rinted by sof
rinted by sof
rinted by sof
rinted by sof
rinted by sof

Jul 9 1999 16:41 swdraw.c	Page 3	Jul 9 1999 16:41
18+4		
) if(!pix(nv6[1]))		fa = 1.00 / a - sin(2.00 * a) / (2 fb = 1.00 / b : sin(2.00 * b) / (2
18++;		c = (a+b)/2.00;
<pre>break; case 1: if(!pix[nv5[1]])</pre>		
18++;		*/ if(fc)
if(!pix(nv6[i]])		if(fa*fc > 0.00)
18++;		a = c; fa = fc;
break; case 2:		else
11.12.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1		, b = c; fb = fc;
)		} alpha2 = (b+a) / 2.00;
15++;		} else
} break;		alpha2 = c;
		} while((b-a) > 0.001);
is /= 2;		$\sin = 0;$
v = (double) iv; $v *= (sqr (3.00) / 2.00);$ $s = (double) is:$		indinusi = indinus = 0; ihplusi = ihplus2 = 0; for (=1; i<=nnogy; i++)
$\begin{array}{lll} & & & & & & & & & & & & & & & & & &$		{ k = (j-1)*nnodx + nnodx/2;
b = 3.14159 / 2.00; fa = 1.00 / a - sin(2.00 * a) / (2.00 *a *a) - coef; fb = 1.00 / b - sin(2.00 * b) / (2.00 *b *b) - coef;		<pre>if(!pix[k-1]) ihminus1 = j; if(!pix[k-2])</pre>
do (1447) - (<pre>ihminus2 = j; if(!pax[k+1]) if(ipax[k+1])</pre>
c = (a+b)/2.00; $fc = 1.00 / c - \sin(2.00 * c) / (2.00 * c * c) - \cos f;$	-	if([pix[k+2]) if(pix[k+2]) ihplus2 = 1:
<pre>printf("%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf</pre>		if(!pix[k]) ih = 1;
if(fc)	-	/* printf("j=%d k=%d pix=%d ih
<pre>if(fa*fc > 0.00) {</pre>		*
a = c; fa = fc;		<pre>if(ihminus1<ih) if="" ih="ihminus1;" im="ihminus1;</pre"></ih)></pre>
else		i (inminus/sin) ii = ihminus2; if (ihminis]
) b = c; fb = fc;		if = inplus; if(ihplus2 <in)< td=""></in)<>
alphal = (b+a) / 2.00;		ih = ihplus2; height = ((double) (nnody - ih)) *
else {		1w = 0; for (1=1; i<=nnodx; i++)
alpha1 = c;		$\begin{cases} k = (nnody-1)*nnodx + i; \end{cases}$
} while((b-a) > 0.001);	,	11(plx[k]) 1w++; /*

Jul 9 1999 16:41 SWGraW.C	Page 4
a = 3.14159 / 2.00; b = 3.14159; fa = 1.00 / a - sin(2.00 * a) / (2.00 *a *a) ~ coef; fb = 1.00 / b - sin(2.00 * b) / (2.00 *b *b) - coef;	
c = (a+b)/2.00; $fc = 1.00 / c - sin(2.00 * c) / (2.00 * c * c) - coef;$	
printf("%lf %lf %lf %lf %lf %lf\n",a,b,c,fa,fb,fc);	
if(fc) if(fa*fc > 0.00)	
{ a = c; fa = fc;	
else	
b = c; $fb = fc;$	
alpha2 = (b+a) / 2.00;	
e1se {	
alpha2 = c;	
<pre>h } while((b-a) > 0.001);</pre>	
ih = 0; ibminus1 = ihminus2 = 0; ihpius1 = ihpius2 = 0; for(j=1; j<=nnody; j++)	
<pre>k = (j-1)*nnodx + nnodx/2; if(!pix(k-1)) if(!pix(k-1)) if(!pix(k-2))</pre>	
<pre>ihminus2 = j; if(!pix[k+1]) if(!pix[k+1]) if(!con)</pre>	
11 (p. 1 k k k k k l l l l l l l l l l l l l l	
/* printf("j=%d k=%d pix=%d ih=%d\n",j,k,pix,ih); */	
<pre>in = i hplus; if (hplus2<in)< td=""><td></td></in)<></pre>	
<pre>iw = 0; for(i=1; i<=nnodx; i++)</pre>	
<pre>{ k = (nnody-1)*nnodx + i; if(pix(k)) iw++;</pre>	

swdraw.c 10

Jul 9 1999 16:41

ecy[3]*n[nv3[1]]+ecy[4]*n[nv6[1]])/3.; break; case 1: doin = 2.*n[1]; gradx[1] = (ecx[1]*n[nv1[1]]+ecx[2]*(doin-n[nv5[1]])+ ecx[3]*(doin-n[nv6[1]])+ecx[4]*n[nv4[1]]+ ecx[5]*n[nv5[1]]+ecx[6]*n[nv6[1]])/3.; break; case 2: doin = 2.*n[1]; gradx[1] = (ecx[1]*n[nv1[1]]+ecx[6]*n[nv6[1]])/3.; doin = 2.*n[1]; gradx[1] = (ecx[1]*n[nv1[1]]+ecx[6]*n[nv6[1]])/3.; doin = 2.*n[1]; break; case 3: doin = 2.*n[1]; gradx[1] = (ecy[1]*n[nv1[1]]+ecx[6]*n[nv2[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv2[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv4[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv4[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv2[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv2[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv2[1]]+ ecy[3]*n[nv3[1]]+ecx[6]*n[nv2[1]]+ ecy[3]*(doin-n[nv2[1]])+ecx[6]*n[nv2[1]])+ ecy[3]*(doin-n[nv2[1]])+ecx[6]*n[nv2[1]])+ ecy[3]*(doin-n[nv2[1]])+ecy[6]*n[nv6[1]])/3.; case 4:	/3.; 5[1]] + (4[1]] +
ak; 11: 2. idx[i] idy[i] idy[i] idy[i] adx[i] adx[i] adx[i] adx[i] adx[i] adx[i] adx[i] adx[i] adx[i]	(3.; (4.[1]) + (4.[1]) + (4.[1]) + (4.[1]) + (4.[1]) + (4.[1]) + (4.[1]) + (4.[1]) + (4.[1]) + (5.[1]) + (6.[1]) /3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.; (7.3.;
ank; 1 = 2. (dy[i] (dy[i] 1 = 2. (dx[i] 1 = 3. (dx[i] 1 = 3. (dx[i] 1 = 3. (dx[i] 1 = 4. (dx[i]	5[1]]}+ (4[1]]+ (5[1]]+ (4[1]]+ (4[1]]+ (4[1]]+ (3.; In-n[nv3[1]])/3.; In-n[nv1[1]]+ v6[1]]/3.; v2[1]]+ (3.; v2[1]]+
1 = 2. 1 2 2 2 2 2 2 2 2 2	5[1]]+ %4[1]+ %5[1]+ %4[1]+ /4[1]+ /3.; in-n[nv3[1]])/3.; in-n[nv1[1]]+ v6[1]]/3.; v6[1]]/3.; v2[1]+ %3.; v2[1]+ %2[1]+
ady(i) 2; 2; 2; 2; 3; 3; 3dx[i] 3dx[i] 3dx[i] 3dy[i] 3dy[i] 3dx[i] 3dx[i] 4; 4;	in-n[nv3[i]]), (3.; in-n[nv3[i]])), (3.; in-n[nv1[i]]), (3.; in-n[nv1[i]]), (3.; v6[i]]), (3.; v2[i]]+ v2[i]]+
ady(i) 2: 2: 2: adx(i) ady(i) 3: in = 2 adx(i) ady(i) 4: 6: 4:	in-n[nv3[i]]), in-n[nv3[i]]), in-n[nv3[i]]), in-n[nv1[i]]), ve[i]]), ve[i]], ve[i]], ve[i]], ve[i]],
adx[1] adx[1] adx[1] adx[1] adx[1] adx[1] adx[1] adx[1] adx[1]	in-n[nv3[i]]))/3.; in-n[nv3[i]]))/3.; in-n[nv1[i]])+ v6[i]])/3.; in-n[nv1[i]])+ v6[i]])/3.; v2[i]]+
2: ln = 2. adx[i] ady[i] 3: ln = 2. adx[i] ady[i]	<pre>in-n[nv3[i]]))/3.; in-n[nv3[i]])/3.; in-n[nv1[i]])+ v6[i]])/3.; v6[i]])/3.; v2[i]]+</pre>
dy[i] dy[i] dy[i] 3: .n = 2. dx[i] dy[i] dx = 4:	in-n[nv3[i]]))/3.; in-n[nv3[i]]))/3.; in-n[nv1[i]])+ v6[i]])/3.; v6[i]])/3.; v2[i]]+
idy[1] sak; .n = 2, idx[1] idx[1] idy[1]	<pre>in-n[nv3[i]])/3.; in-n[nv3[i]])/3.; in-n[nv1[i]])+ v6[i]])/3.; in-n[nv1[i]])+ v6[i]]/3.; v2[i]]+</pre>
38k; 31 = 2. 32k[1] 34k;	in-n[nv1[i]])+ v6[i]])/3.; v6[i]])/3.; v2[i]]+
3: .n = 2.*n .dx[i] = .dy[i] = .ak;	in-n[nv1[i]])+ v6[i]]/3.; in-n[nv1[i]])+ v6[i]]/3.; v2[i]]+
idx[i] = idy[i] = sak;	in-n[nv1[i]]) + v6[i]])/3.; in-n[nv1[i]]) + v6[i]])/3.; v2[i]] +
ndy[i] =	in-n[nul[i]]) + v6[i]]) / 3.; v2[i]] +
4:	v2[1]]+ /3.; v2[1]]+
Case 4:	v2[1]]+ /3.; v2[1]]+
	v2[1]]+ /3.; v2[1]]+
[i] =	/3.; v2[1]]+
<pre>gex[6]*n[nv5[1]]*ecx[6]*n[nv6[1]]*/3.; grady[1] = (ecy[1]*(doin.n[nv4[1]])*ecy[2]*n[nv2[1]]*</pre>	
ecy[5]*n[nv5[1]]+ecy[4]*n[nv4[1]]+ ecy[5]*n[nv5[1]]+ecy[6]*n[nv6[1]])/3.;	/3.;
break;	
E	
gradx[i] = (ecx[l]*n[nvl[i]]+ecx[2]*n[nv2[i]]+ ecx[3]*n[nv3[i]]+ecx[4]*(doin-n[nvl[i]])	11[1]])+
<pre>grady[1] = (ecy[1]*n[nv1[1]]+ecy[2]*n[nv2[1]]+</pre>	/3.; 1(1]])+
ecy[5]*n[nv5[1]]+ecy[6]*n[nv6[1]])/3.; break;	/3.;
<pre>doin = 2.*n[1]; gradx[1] = (ecx[1]*(doin-n[nv4[i]))+ecx[2]*(doin-n[rgrafi])+ecx[2]*(doin-n[rgrafi</pre>	in-n[nv5[i]])+
<pre>grady[1] = (ecy[1])*(doin-n[nv3[1]))/3.; grady[1] = (ecy[1])*(doin-n[nv4[1])*ecy[2]*(doin-n[nv5[1])) ecy[3]*n[nv3[1]]*ecy[4]*n[nv4[1]]*</pre>	3(1]]))/3.; in-n[nv5[i]])+
ecy[5]*n[nv5[1]]+ecy[6]*(doin-n[nv6[1]])	·6[i]]))/3.;
Case 7:	
n = 2.*n $dx[i] =$	5[1]])+
<pre>ecx[3]*(doin-n[nv6[i])) + ecx[4]*(doin-n[nv1[i]))+ ecx[5]*n[nv1[i] + ecx[6]*n[nv6[i])) / 3.; grady[i] = (ecy[1]*n[nv1[i]] + ecy[2]*(doin-n[nv6[i])) + ecy[3]*(doin-n[nv6[i]) + ecy[4]*(doin-n[nv1[i]]) +</pre>	in-n[nv1[1]])+ 3.; 5[1])+ in-n[nv1[1]])+
ecy[5]*n[nv5[i]]+ecy[6]*n[nv6[i]])/3.;	/3.;
gradx[i] = (ecx[1]*(doin-n[nv4[i]))+ecx[2]*(doin-n[nv5[i]))+	in-n[nv5[i]])+

swdraw.c

Ţ

swdraw.c

#	1
긋	ı
flumag3.mec	١
Ξ	١
ස	ı
g	ı
≥	ı
₹	ı
5	ı
=	1
eg	١
5	ı
Printed by sofonea from fl	ı
S	١
õ	ı
8	ı
Ĕ	I
듄	ı
_	ı
	I
	ı
	ł
	ı
	l
	ı
	ı
	ı
	ı
	ı
	ı
	1

Jul 9 1999 16:41 SWdraw.c	Page 7	
<pre>ecx[4]*n[nv4[1]]+ecx[5]*n[nv5[1]])/3.; grady[1] = (ecy[1]*(doin-n[nv4[1])+ecy[2]*(doin-n[nv5[1]])+ ecy[4]*n[nv4[1]]+ecy[5]*n[nv5[1]])/3.;</pre>		
<pre>break; case 9: doin = 2.*n[i]; gradx[i] = (ecx[1]*n[nv1[i]]+ecx[2]*n[nv2[i]]+ ecx[4]*(doin-n[nv1[i])+ecx[5]*(doin-n[nv2[i]))/3.; cyadv[i] = (ecv[1]*n[nv1[i]]+ecv[2]*n[nv2[i]]+</pre>		
<pre>break; break 10:</pre>		
<pre>grady[i] = (cc[3]*n[nv3[i]]+ecx[4]*n[nv4[i]]+</pre>		
break; 		
i; ii; bile doipetrel = 2./3.; (i=1; i<=nnodes; i++) switch (boundary_mode[1])		
<pre>case 0: nlap[i] = doipetrei*(n[nv1[i]]+n[nv2[i]]+n[nv3[i]]+ n[nv4[i]]+n[nv5[i]]+n[nv6[i]] -6.*n[i]);</pre>		
<pre>case 1: nlap[i] = doipetrei*(n[nv1[i]]+n[nv4[i]]-2.*n[i]); break;</pre>		
<pre>case 2:</pre>		
case : nlap[i] = 0.; break;		
<pre>case 4: nlap[i] = doipetrei*(n[nv2[i]]+n[nv3[i]]+ n[nv5[i]]+n[nv6[i]]-4.*n[i]);</pre>		
<pre>break; case 5: nlap[i] = doipetrei*(n[nv2[i]]+n[nv3[i]]+ nlap[i] = doipetrei*(n[nv2[i]]+n[nv3[i]]+</pre>		
break; case 6: nlanf4] = 0.:		
n.ap[1] = 0.; break; case 8:		
nlap[1] = 0.; break;		
case 9: nlap[i] = 0.; break:		
case 10: nlap[i] = 0.; break;		

swweep.c

Jul 4 1999 16:24

Page 2

Jul 4 1999 16:24 Swweep.c	rage I
<pre>#include <stdio.h> #include <string.h> #include <stdlib.h> #include <math.h></math.h></stdlib.h></string.h></stdio.h></pre>	
<pre>#include "swhead.h" double f0[], double f1[], double f2[], double f3[],</pre>	
<pre>void automaton_aux(double nf0[], double nf1[], double nf2[], double nf4[],</pre>	
<pre>{ FILE *frez; int i, j, k; double uxloc, uyloc, uxxloc, uyyloc, uloc2; double cazero, ca, cb, cczero, cc, cd, cgxx, cgxy, xgyx, cgyy; double feq(7); double fx, fy;</pre>	
<pre>for(i=1; i<=nnodes; i++) {</pre>	
if(nf0[1] < 0.00)	
printf("	
4	
} if (nf1[1] < 0.00)	
{ printf("i=%d nf2=%lf\n",i,nf2[i]);	
if (nf3[i] < 0.00)	
printf("	
1f(nf4[i] < 0.00) {	
1f(nf5[4] < 0.00)	
printf("	
<pre>if jnf6[1] < 0.00)</pre>	
<pre>printf("1=%d nf6=%lf\n",i,nf6[1]); }</pre>	
floc[i] = nf0[i]+nf1[i]+nf2[i]+nf3[i]+nf4[i]+nf5[i]+nf6[i];	
<pre></pre>	
{ for (i=1; i<=nnodes; i++)	
٠	

-
<pre>case 0: edip[i] = floc[nvl[i]] * (mm-3.7.* (nm*ecx[1]+my*ecy[1]) +</pre>
<pre>LIOC[INV2[1]] * (Imm-3.0* (mx*ecx[2]+my*ecy[2])* (mx*ecx[2]+my*ecy[2])) + floc[nv3[1]] *</pre>
$(mm-3.0^{\circ}(4)+my*ecy[4])*(mx*ecx[4]+my*ecy[4])$ + floc[ny5[1]] *
<pre>[mm-3.0*(mx*ecx[5]+my*ecy[5])*(mx*ecx[5]+my*ecy[5])) + flociny6[1]] *</pre>
case 1: catp[1] = floc[nv1[1]] * (mn-3.0*(mx*ecx[1]+my*ecy[1])*(mx*ecx[1]+my*ecy[1])) +
<pre>floc[nv4[1]] * (mm-3.0*(mx*ecx[4]+my*ecy[4])*(mx*ecx[4]+my*ecy[4])) +</pre>
floc[ny3 1,] * (mx*ecx[5]+my*ecy[5])*(mx*ecx[5]+my*ecy[5])) +
linc(involil) " (mm-3.0*(mx*ecx[6]+my*ecy[6])*(mx*ecx[6]+my*ecy[6])); break;
] = flo 3.0*(mx
floc[nv2[1]] * (mm.*ecx[2]+my*ecy[2])*(mx*ecx[2]+my*ecy[2])) +
<pre>i_LOC[Nv3[1]</pre>
} edip[i] *= floc[i];
gradient (edip, egradx, egrady);
for(i=1; i<=nnodes; i++)
<pre>' uxloc = (nf1[i]*ecx[1]+nf2[i]*ecx[2]+nf3[i]*ecx[3]+ nf4[i]*ecx[4]+nf5[i]*ecx[6]+nf6[i]*ecx[6]) floc[i]; uyloc = (nf1[i]*ecy[i]+nf2[i]*ecy[5]+nf3[i]*ecy[3]+ nf4[i]*ecy[4]+nf5[i]*ecy[5]+nf6[i]*ecy[6])/floc[i];</pre>
uloc2 = uxloc*uxloc + uyloc*uyloc;
<pre>ca = (floc[1]*temp) / (1.0 - floc[1]*bpsi) - apsi * floc[1] * floc[1] - kappa * floc[1] * flap[1];</pre>
<pre>cazero = floc[i] - 2.00 * ca; ca /= 3.00; cb = floc[i]/3; cc = -floc[i]/6; cc = cloc[i]/6; cc = 2.*floc[i]/3; cd = 2.*floc[i]/3; cgxx = kappa*(fgradx[i]*fgradx[i]-fgrady[i]/3.0; cgxy = -cgxx [0.0*kappa*fgradx[i]*fgrady[i]/3.;</pre>
<pre>uxxloc = uxloc*uxloc; uxyloc = uxloc*uyloc; uyyloc = uyloc*uyloc;</pre>
feq[0] = cazero + cczero * uloc2;
if(feq[0] < 0.00)

swweep.c 13

swweep.c

Jul 4 1999 16:24

for (j=1; j<=6; j++)
if(floc[1] > 3.5) /* liquid */
feq[j] = ca + cb*(ecx[j)*uxloc+ecy[j]*uyloc) + cc*uloc2 +
<pre>fx*ecx[j] + fy*ecy[j] + cd*(uxxloc*ecxy[j]+2.*uxyloc*ecxy[j]+</pre>
<pre>+ ayxe(eceyy[j])+egxy*ecxy[j] - tau3 * floc[i] * gradmiulwall2 * ecy[j];</pre>
else /* vapor */
<pre>leq(j) = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) + cc*uloc2 +</pre>
<pre>fx*ecx[j] + fy*ecy[j] +</pre>
uyyloc*ecyy[j]) + cgx**(ecxx[j]-ecyy[j])+cgxy*ecxy[j]- tau3 * floc[i] * gradmiu2wall2 * ecy[j];
} break;
break; case 3: /* dipolar interaction */
<pre>switch(boundary_mode[i]) {</pre>
<pre>case 0:</pre>
Dreak; case 1: if(floc[i] > 3.5) /* liquid */
<pre>{ tx = tau3 * floc[i] * egradx[i]; fy = tau3 * floc[i] * (gradminlwall1 * ecy[i] + egrady[i]);</pre>
else 1
<pre>fx = tau3 * floc[i] * egradx[i]; fy = tau3 * floc[i] * (gradmiu2wall1 * ecy[i] + egrady[i]);</pre>
<pre>break; case 2: if(floc[i] > 3.5) /* liquid */</pre>
<pre>{</pre>
else
<pre>fx = tau3 * floc[i] * egradx[i]; fy = tau3 * floc[i] * (gradmiu2wall2 * ecy[i] + egrady[i]);</pre>
break;
for (j=1; j<=6; j++)
<pre>feq[j] = ca + cb*(ecx[j]*uxloc+ecy[j]*uyloc) + cc*uloc2 + cd*(uxxloc*ecxx[j]+2.*uxyloc*ecxy[j]+uyyloc*ecyy[j]) + cqxx*(ecxx[j]-ecyy*ecxy[j] - fx * ecxt[j]-ecyv*ecxy[j] - fx * ecxt[j] - fv * ecv[j];</pre>
break;
nff0[i] = nf0[i] - (nf0[i]-feq[0])/tau;

swweep.c

Cunal Contract		
	-	
		 0000

Jul 4 1999 16:24	### TEET [[1 1 1 1 1 1 1 1 1	46	ff3[1] = f ff3[1] = f broak; case 10: ff2[nv2[1] ff2[nv3[1] ff3[nv3[1] ff2[1] = f ff2[1] = f ff3[1] = f		<pre>void automaton(void) int autiter; autiter = 0; while(autiter < r</pre>	
Jul 4 1999 16:24 Swweep.c	<pre>switch(boundary_mode[1]) case 0: nff[[nu1[1]] = nff[1] - (nff[1]-feq[1])/tau; nff2[nv2[1]] = nff1[1] - (nf4[1]-feq[1])/tau; nff2[nv2[1]] = nff2[1] - (nf2[1]-feq[2])/tau; nff2[nv3[1]] = nff2[1] - (nff2[1]-feq[3])/tau; nff6[nv3[1]] = nff1[1] - (nff2[1]-feq[3])/tau; nff6[nv6[1]] = nff1[1] - (nff[1]-feq[1])/tau; nfff[nv4[1]] = nff1[1] - (nff1[1]-feq[1])/tau; nff5[1] = nff2[1] - (nff1[1]-feq[1])/tau; nff6[1] = nff2[1] - (nff1[1]-feq[2])/tau; nff6[nv6[1]] = nff1[1] - (nff1[1]-feq[3])/tau; nff2[nv4[1]] = nff1[1] - (nff1[1]-feq[3])/tau; nff2[nv2[1]] = nff1[1] - (nff1[1]-feq[3])/tau; nff2[n] = nff1[1] - (nff1[1]-feq[3])/tau; nff2[1] = nff1[1] - (nff1[1]-feq[1])/tau; nff1[1] nff1[1] - (nff1[1]-feq[1])/tau</pre>	/* case 3: ffl[nv1[i]] = f1[i] - (f1[i]-feq[i])/tau; ff2[nv2[i]] = f2[i] - (f2[i]-feq[2])/tau; ff6[nv6[i]] = f6[i] - (f6[i]-feq[6])/tau; ff1[i] = f4[i] - (f4[i]-feq[4])/tau; ff2[i] = f5[i] - (f5[i]-feq[5])/tau; break;	case 4: Iff[nv4[1]] = f4[1] - (f4[1]-feq[4])/tau; Iff[nv4[1]] = f2[1] - (f2[1]-feq[2])/tau; Iff2[nv3[1]] = f3[1] - (f2[1]-feq[3])/tau; Iff5[nv5[1]] = f5[1] - (f5[1]-feq[5])/tau; Iff4[1] = f1[1] - (f5[1]-feq[6])/tau; Iff(ff0[1] o.00) (ff1] - feq[1])/tau; (ff4[1] < 0.00) (ff1] - feq[0])/tau; (ff4[1] < 0.00) (ff1] - feq[0])/tau; (ff4[1] < 0.00) (ff2[1] < 0	fprint(frez, "wmiux[i] = 20.16f wmiuy[i] = \$20.16f\n fx=\$20.16f nfy=\$20.16f\nfx=\$20.16f fy=\$20.16f\n", wmiux[i],wmiuy[i],nloc[i] *force_x,nloc[i] *force_y,fx,fy);	<pre>fprintf(frez,"ecx=%f %f %f %f %f %f\n",ecx[1],ecx[2],ecx[3],</pre>	break; case 5:

Jul 4 1999 16:24 swweep.c	Page 6
(f6[1]-feq[6])/fau; (f2[1]-feq[1])/fau; (f2[1]-feq[1])/fau; [f2[1]-feq[1])/fau; [f1]-(f1]-feq[1])/fau; (f4[1]-feq[4])/fau; (f2[1]-feq[3])/fau; (f2[1]-feq[3])/fau;	
4[1] - (f4[1]-feq[4]) 5[1] - (f5[1]-feq[5]) (f6[1]-feq[6])/tau, (f1[1]-feq[1])/tau, - (f2[1]-feq[2])/tau, - (f3[1]-feq[3])/tau,	
= II[1] = f2[i] [] - (f4 [] - (f6 [] - (f6	
<pre>ff2[nv2[1]] = f2[1] - (f2[1]-feq[2])/tau; ff3[nv3[1]] = f3[1] - (f3[1]-feq[3])/tau; ff4[nv4[1]] = f4[1] - (f4[1]-feq[4])/tau; ff2[1] = f5[1] - (f5[1]-feq[5])/tau; ff3[1] = f5[1] - (f5[1]-feq[5])/tau; ff4[1] = f1[1] - (f1[1]-feq[1])/tau; */ break;</pre>	
} }	
<pre>int autiter; int autiter; autiter = 0; while autiter < niter_cycle)</pre>	

May 13 1997 20:19	swmain.c	Page 1
<pre>#define MAIN_HEADER #include <stdio.h> #include <string.h></string.h></stdio.h></pre>		
#include "swhead.h"		
<pre>void sw_input(void); void sw_output(void); void build_names(int); void detavec_hex(void); void xv(double n0[], double n void init_arrays_aux(void); void init_arrays_aux(void); void automaton(void);</pre>	ble n1[], double n2[], double n3[], ble n5[], double n6[], char filename[]); d);	
<pre>void main(void) { int isim, isav, icycle; FILE *fin, *fout, *fsav;</pre>	,	
<pre>sw_input(); sw_output();</pre>		
<pre>for(isim=0; isim<nsim; isim++)="" pre="" {<=""></nsim;></pre>	isim++)	200
<pre>switch(akey_status[isim])</pre>	[isim])	
case 0: build_names(isim); getavec_hex(); key_init = akey_init(isim]; key_interaction = akey_interaction = a	see 0; bould_names(isim); getavec_hex(); getavec_hex(); key_init = akey_init(isim); key_interaction = akey_interaction_ini[isim]; key_interaction = akey_boundary_ani(isim); temp_in = atemp_in[isim]; kappa = akappa[isim]; ax = axx[isim]; ax = axx[isim]; mx = anx[isim]; mx = anx[isim]; mx = anx[isim]; mx = anx[isim]; gradmiulvall1 = agradmiulvall1[isim]; gradmiulvall2 = agradmiulvall2[isim]; gradmiulvall2 = agradmiulvall2[isim]; init_arrays_hex(); nite_rinit = aniter_init(isim); nite_rinit = aniter_init(isim);	
<pre>temp = temp_in; iter = 0; xv(f0,f1,f2,f3, /* profile(f0,f1,f automaton();</pre>	<pre>temp = temp_in; tter = 0; xv(f0,f1,f2,f3,f4,f5,f6,xv_name); xv(f0,f1,f2,f3,f4,f5,f6,profile_name); */ automaton();</pre>	
ter = 0,0,1 (f. f. f	<pre>iter = 0.0.v(f), iter = 0.0.v(f), f1,f2,f3,f4,f5,f6,xv_name); minko(); profile(f0,f1,f2,f3,f4,f5,f6,profile_name); */ profile(f0,f1,f2,f3,f4,f5,f6,profile_name); */ profile(f0,f1,f2,f3,f4,f5,f6,profile_name); */ key_interaction = akey_interaction_fin(isim); key_boundary = akey_boundary_fin(isim); inti_arrays_hex_aux(); inti_arrays_hex_aux(); inti_arrays_les(isim); ncycles = ancycles(isim); inter_cycle = aniter_cycle(isim);</pre>	
•		

for(issav-0; issavchsav; issavch) for(issav-0; ispavchsav; issavch) print('ssavch') print('ssavch') automaton(); xv(fo, fi, f2, f2, f4, f5, f6, profile_name); profile(f0, fi, f2, f3, f4, f5, f6, profile_name); } profile(f0, fi, f2, f3, f4, f5, f6, profile_name); }	May 13 1997 20:19	swmain.c	Page 2
print("tsav**d ioyole-#d\n",isav,icycle); xv(fo,fi,fi,fi,fi); minko(); profile(f0,fi,fi); profile(f0,fi,fi); }	for (isav=0 {	; isav <nsav; isav++)<="" td=""><td></td></nsav;>	
xv(f0, f1, f2, f3, f4, f5, f6, xv_name); minko(); profile(f0, f1, f2, f3, f4, f5, f6, profile_name); } } }		cycle);	
	xx mi pr pr }		

16

Page 1					
sw.job					
	8.00				
œ	0.0094 .00 0.0				
99 16:4	1000 0.550 000000 0.01				
Jul 17 1999 16:48	1 0 Drop-mx 10 1 100 1000 70 2 0 1 3.5 0.550 0.550 0.0094 0.8 0.00 0.0000000 0.00 0.00 0.00 0.00		,	•	
ನ	понгтоо	 			

doi w

17